A Big Kid's Playground: what can you do with linear algebra, differential equations, and thousands of processors?

Van Emden Henson

Center for Applied Scientific Computing

Lawrence Livermore National Laboratory

vhenson@llnl.gov

http://www.casc.gov/CASC/people/henson

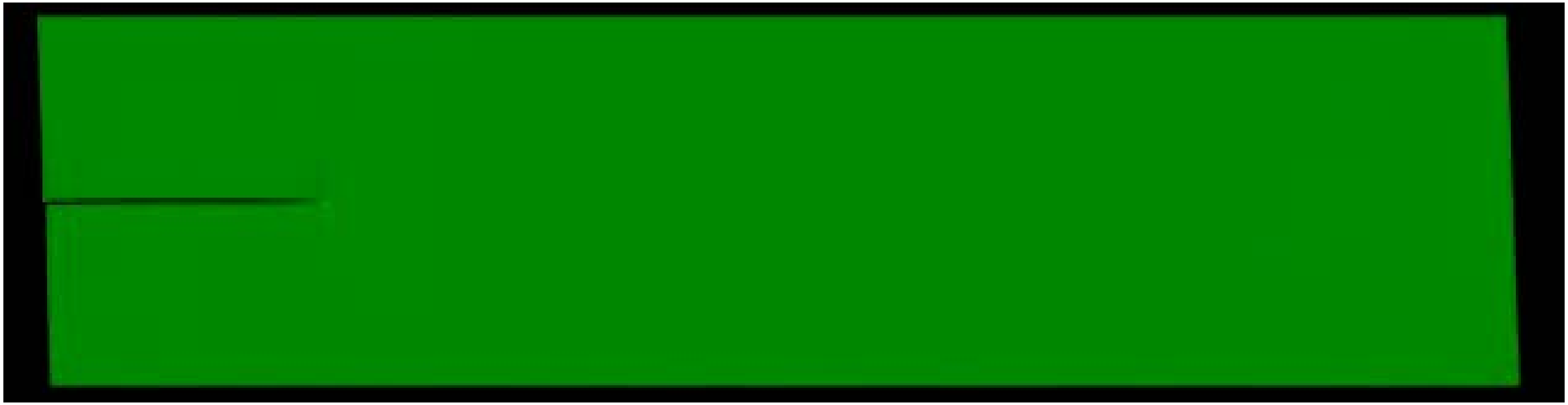
This work was performed, in part, under the auspices of the United States Department of Energy by University of California Lawrence Livermore National Laboratory under contract number W-7405-Eng-48.

$Ax = b$

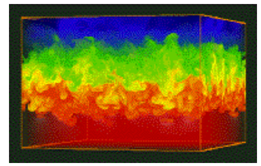
- Okay, I know how to solve this equation: give me b and I'll give you x.
- But what can I do with that knowledge?

Example: Fracture dynamics

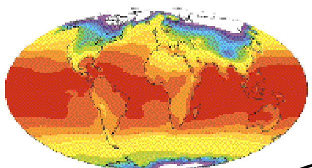
- New simulations show it is possible for shear-induced cracks to travel at transsonic and supersonic speeds under certain conditions, contradicting classical theory.
- This movie shows the crack propagation in the harmonic (linear) material. The speed in this case is transsonic after the emergence of the daughter crack
- This simulation was performed on several thousand processors of the ASCI White supercomputer at LLNL

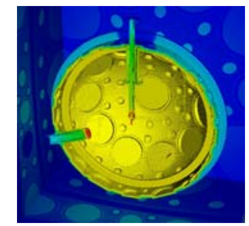


Simulation is emerging as a peer to theory and experiment



Experiments are prohibited





SSP / ASCI
radiation transport
hydrodynamics

Environmental
global climate
groundwater flow

Engineering
structural dynamics
electromagnetics

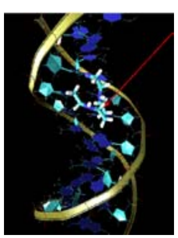
Experiments are impractical

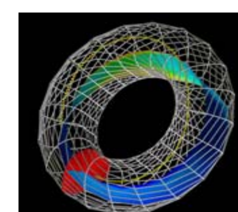
Experiments are too expensive

Physics & Biology
materials modeling
drug design

Enabling Computational Technologies
computational math
computer science
MPP

Lasers & Energy
combustion
ICF modeling





Large-scale simulation plays a central role in nearly every LLNL program

How does *Ax=b* arise? A simple example

- We derive, over the next several slides, a discretization of the heat equation

$$u_t(x,y,t) = \kappa \nabla^2 u(x,y,t) + f(x,y,t)$$

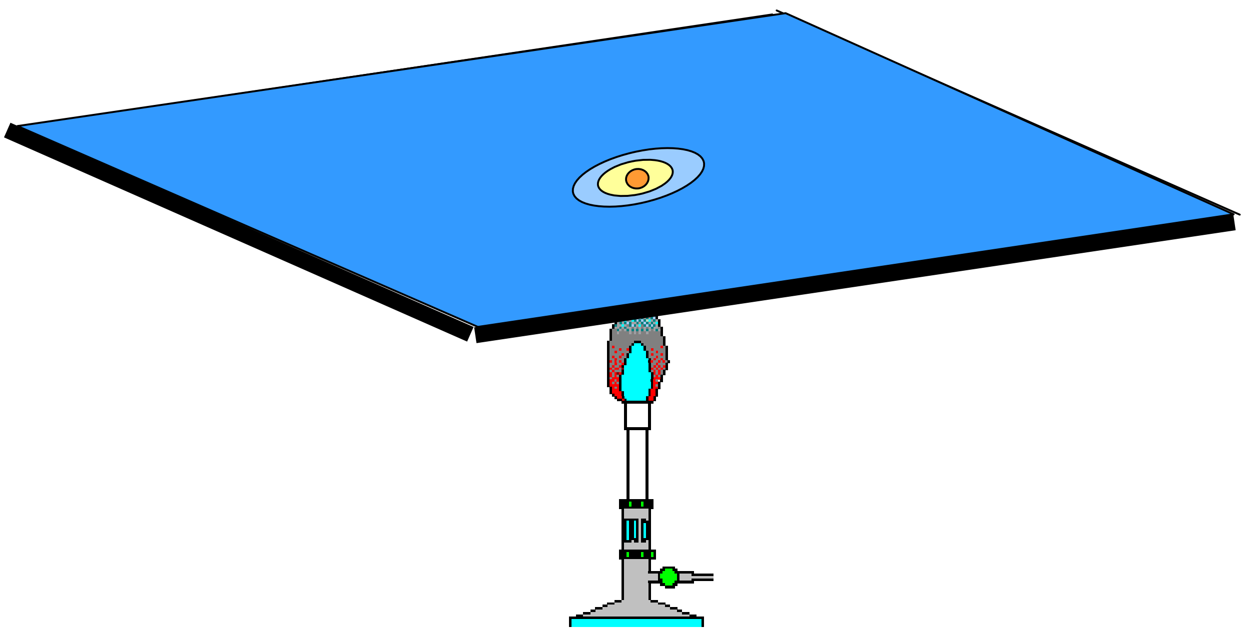
to arrive at the sequence of problems of the form

$$Ax^k = b^k, \quad k = 0, 1, 2, \ldots$$

where each step entails a matrix solve

A simple example

- Consider a rectangular, homogeneous metal plate. Suppose the four edges of the plate are held at a constant temperature of 0, and a constant source of heat is placed in the middle of the plate:



The heat equation

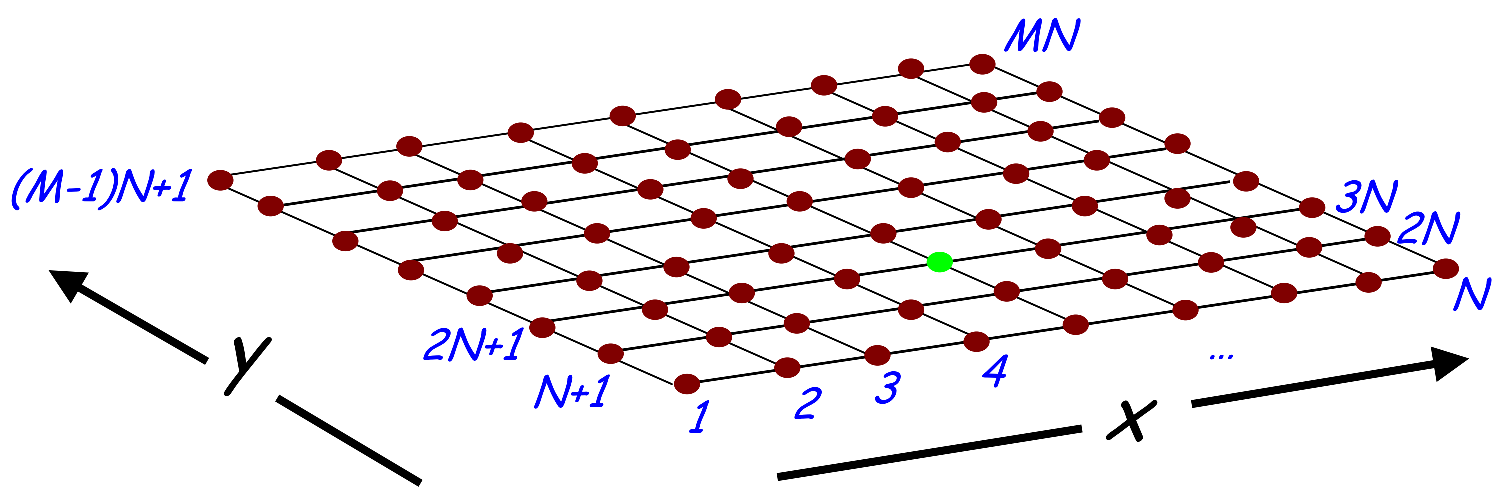
- The temperature at time t and location *(x,y)* in the plate, denoted *u(x,y,t)*, is determined by the partial differential equation (the heat equation):

$$\underbrace{\frac{\partial u(x,y,t)}{\partial t}}_{\text{Rate of change of temperature}} = \kappa \overbrace{\left(\frac{\partial^2 u(x,y,t)}{\partial x^2} + \frac{\partial^2 u(x,y,t)}{\partial y^2}\right)}^{\text{Laplacian, i.e., curvature or rate of change of rate of change in x and y directions}} + \underbrace{f(x,y,t)}_{\text{Source term (bunsen burner)}}$$

Thermal diffusivity

Discretizing the problem

- Approximate *u(x,y,t)* at the intersections of gridlines
- *N* gridlines in the *x*-direction, *M* in the y-direction
- Since *u=0* on boundaries, we don't show boundaries
- Let *h* be the grid spacing (equal in both directions)
- A gridpoint *(i,j)* is located at $(x_i=ih,\ y_j=jh)$
- We number the *MN* gridpoints lexographically [example: if *N=9, M=8* as shown, point 23 is located at *(i,j)=(3,5)*]



We use Taylor series derive approximate derivative formulae

$$g(x+\Delta x)=g(x)+\Delta x\frac{dg(x)}{dx}+\frac{(\Delta x)^2}{2!}\frac{d^2g(x)}{dx^2}+\frac{(\Delta x)^3}{3!}\frac{d^3g(x)}{dx^3}+\frac{(\Delta x)^4}{4!}\frac{d^4g(\xi)}{dx^4}$$

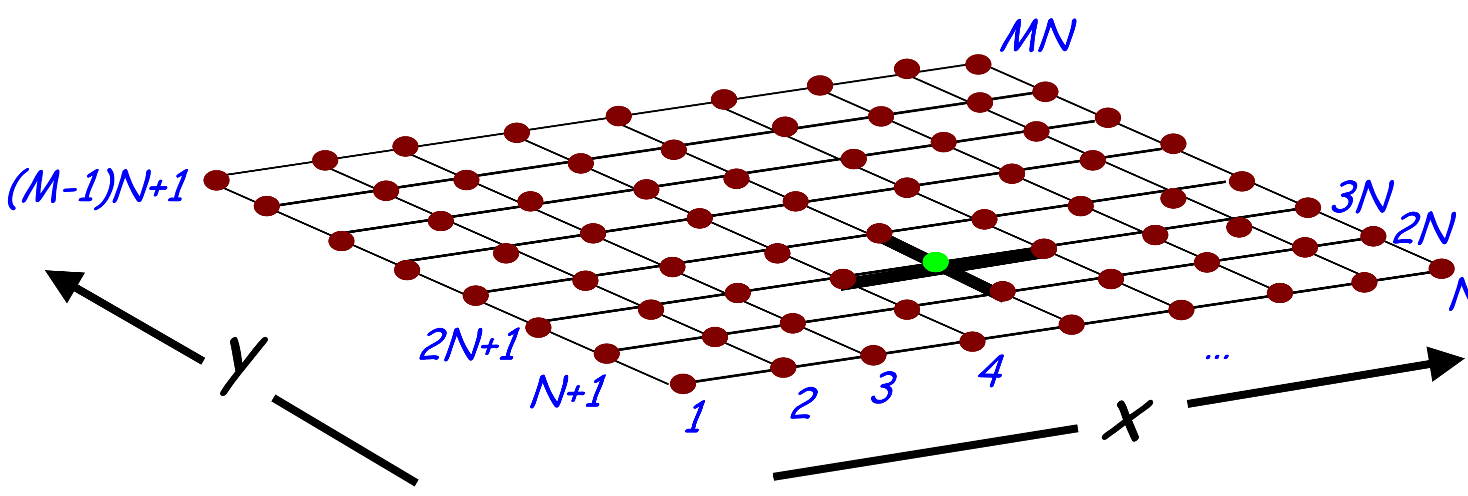
$$g(x-\Delta x)=g(x)-\Delta x\frac{dg(x)}{dx}+\frac{(\Delta x)^2}{2!}\frac{d^2g(x)}{dx^2}-\frac{(\Delta x)^3}{3!}\frac{d^3g(x)}{dx^3}+\frac{(\Delta x)^4}{4!}\frac{d^4g(\varsigma)}{dx^4}$$

$$g(x+\Delta x)+g(x-\Delta x)=2g(x)+(\Delta x)^2\frac{d^2g(x)}{dx^2}+\frac{(\Delta x)^4}{12}\frac{d^4\left(g(\xi)+g(\varsigma)\right)}{dx^4}$$

$$\boxed{\frac{g(x+\Delta x)-2g(x)+g(x-\Delta x)}{(\Delta x)^2}=\frac{d^2g(x)}{dx^2}+O\left((\Delta x)^2\right)}$$

- We use the notation $u_{i,j}^{k} = u(x_i, y_j, t_k)$
- Use Taylor series to approximate partial derivatives at *(i,j)* and time *kΔt*:

$$\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)_{i,j}^{k} = \frac{u_{i+1,j}^{k} - 2u_{i,j}^{k} + u_{i-1,j}^{k}}{h^2} + \frac{u_{i,j+1}^{k} - 2u_{i,j}^{k} + u_{i,j-1}^{k}}{h^2} + O(h^2)$$



- We can use the same approach to obtain an approximation for the time derivative. From the Taylor series:

$$g(x-\Delta x) = g(x) - \Delta x \frac{dg(x)}{dx} + \frac{(\Delta x)^2}{2!}\frac{d^2u(\zeta)}{dx^2}$$

- Rearranging:

$$\frac{g(x) - g(x-\Delta x)}{\Delta x} = \frac{dg(x)}{dx} + O\left(\Delta x\right)$$

- Applying this to the partial derivative in the heat equation yields

$$\frac{\partial u(x,y,t)}{\partial t} = \frac{u_{i,j}^{k} - u_{i,j}^{k-1}}{\Delta t} + O\left(\Delta t\right)$$

- We drop the remainder terms and substitute into the heat equation. Assuming that we know temperature at *(i,j)* and time *k-1*, we obtain a system of equations for the temperature at time *k*, for all *i=1,2,…N* and *j=1,2,…M* .

$$\frac{u_{i,j}^{k}-u_{i,j}^{k-1}}{\Delta t}=\kappa\frac{-4u_{i,j}^{k}+u_{i-1,j}^{k}+u_{i+1,j}^{k}+u_{i,j-1}^{k}+u_{i,j+1}^{k}}{h^{2}}+f_{i,j}^{k}$$

Note that when *i=1*, the equation calls for $u_{0,j}^{k}$ that *i=N*, requires $u_{N+1,j}^{k}$. Similarly, *j=1* needs $u_{i,0}^{k}$ and *j=M* entails $u_{i,M+1}^{k}$.

But these are all boundary values, and by the boundary condition *u=0* vanish from the equations.

- Collect all the unknowns on the left and the known quantities on the right, we obtain

$$(1+4r)u_{i,j}^{k}+r\left(u_{i-1,j}^{k}+u_{i+1,j}^{k}+u_{i,j-1}^{k}+u_{i,j+1}^{k}\right)=F_{i,j}^{k}$$

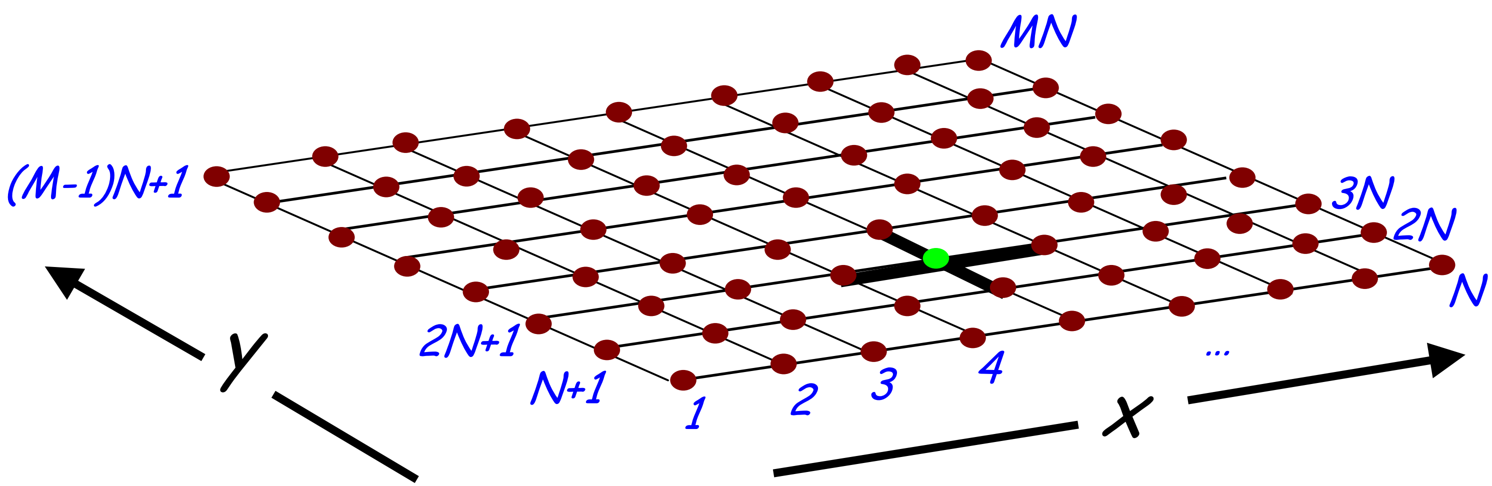
where $r=\dfrac{\kappa\Delta t}{h^2}$ and $F_{i,j}^{k}=(\Delta t)f_{i,j}^{k}+u_{i,j}^{k-1}$

- We have *MN* equations in *MN* unknowns to approximate the temperature *u(x,y,kΔt)* if we know the temperature *u(x,y,(k-1)Δt)* .

- We order the *MN* unknowns into a vector of length *MN*, and organize the equations to match.

- We have *MN* equations in *MN* unknowns to approximate the temperature *u(x,y,kΔt)* if we know the temperature *u(x,y,(k-1)Δt)* .

$$
\begin{pmatrix}
C & B & & & & \\
B & C & B & & & \\
 & B & C & B & & \\
 & & B & C & \ddots & \\
 & & & \ddots & \ddots & B \\
 & & & & B & C
\end{pmatrix}
\begin{pmatrix}
U^k_{1\to N} \\
U^k_{N+1\to 2N} \\
U^k_{2N+1\to 3N} \\
U^k_{3N+1\to 4N} \\
\vdots \\
U^k_{(M-1)N+1\to MN}
\end{pmatrix}
=
\begin{pmatrix}
F^k_{1\to N} \\
F^k_{N+1\to 2N} \\
F^k_{2N+1\to 3N} \\
F^k_{3N+1\to 4N} \\
\vdots \\
F^k_{(M-1)N+1\to MN}
\end{pmatrix}
$$



- The matrices *C* and *B* are *NxN* and are given by:

$$C = \begin{pmatrix} 1+4r & r & & & \\ r & 1+4r & r & & \\ & r & 1+4r & r & \\ & & \ddots & \ddots & \ddots \\ & & & r & 1+4r \end{pmatrix} \qquad B = rI$$

- Letting *A* be the block tridiagonal matrix with blocks *B* & *C*, the temperature at any time $k\Delta t$ is computed from the temperature at time $(k-1)\Delta t$ by solving the matrix equation

$$AU^k = F^k$$

- Thus, we may start with an initial temperature distribution U^0 and solve the sequence of problems, each having the same matrix A and a new right-hand side F^k.

$$
\begin{array}{ll}
U^1 = A^{-1}F^1 & F^2 = (\Delta t) f^2 + U^1 \\
U^2 = A^{-1}F^2 & F^3 = (\Delta t) f^3 + U^2 \\
\vdots \quad\quad \vdots & \vdots \quad\quad\quad \vdots \\
U^k = A^{-1}F^k &
\end{array}
$$

- The quality of the solution depends on the truncation errors h and Δt.

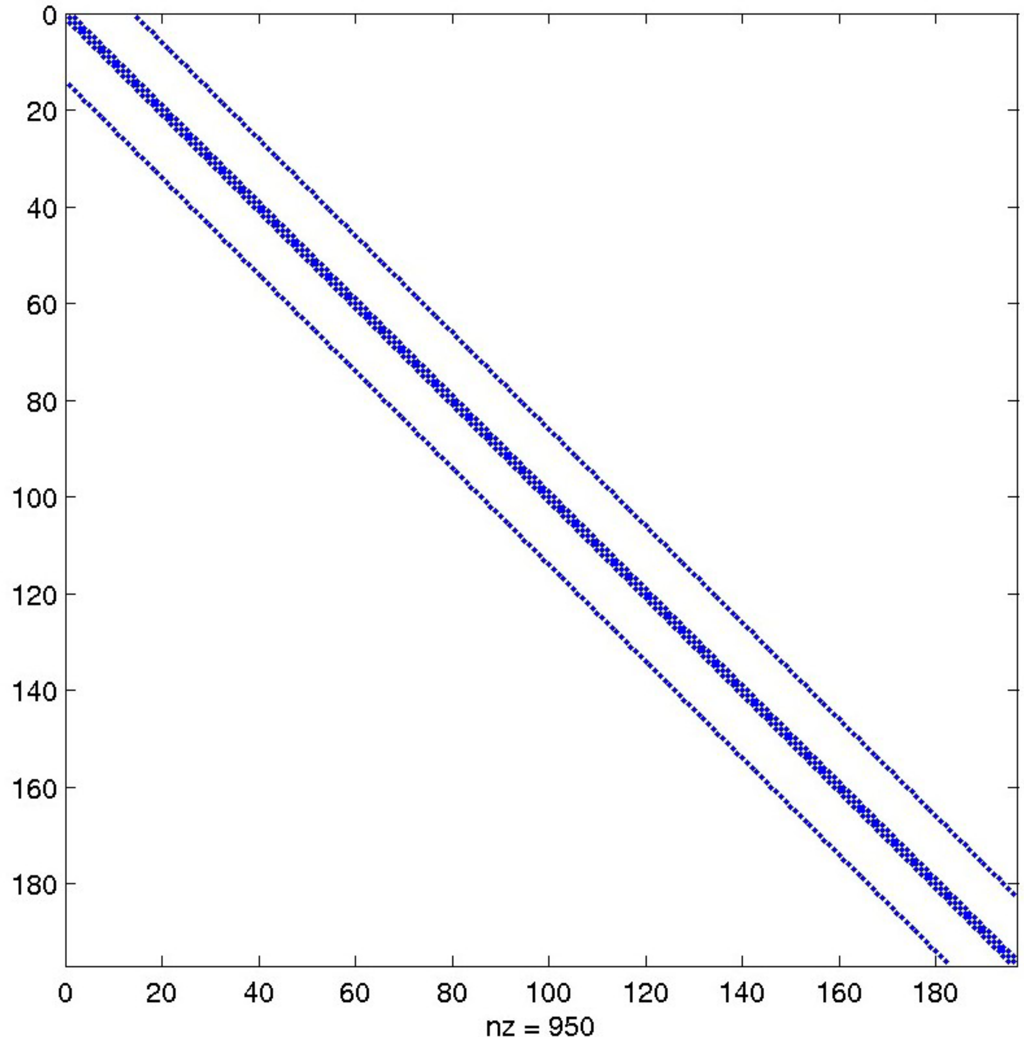
- Note: This particular method, fully implicit backward-in-time, is derived because it is easy to explain in brief. This is NOT a robust method for this problem.

- The big question: How do we solve

$$AU^k = F^k \quad ?$$

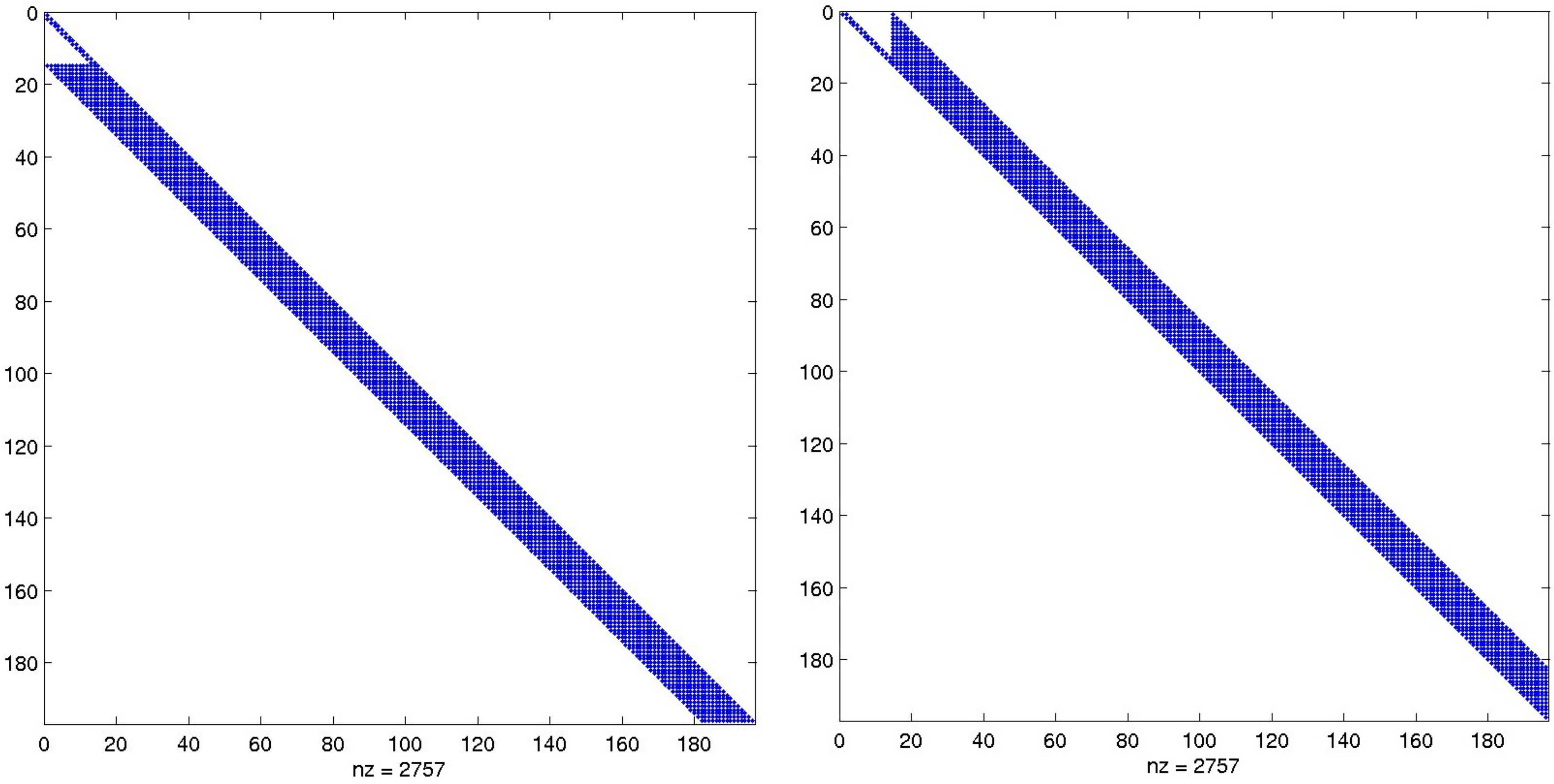
How about Gaussian elimination (i.e., LU factorization)?

- The nonzero structure of A (14x14 grid, MN=196) is:



How about Gaussian elimination (i.e., LU factorization)?

- The nonzero structures of L and U are:



Of course, it's never this simple

- Numerical stability & accuracy requirements dictate the sizes of Δt and h

- Anisotropic diffusion coefficients: commonly the diffusivity is spatially varying, in which case the equation becomes

$$u_t(x,y,t) = \nabla \bullet \left(\kappa(x,y) \nabla u(x,y,t) \right) + f(x,y,t)$$

- Often, the simulation is on a domain whose physical properties are changing over time; that is, A is a time-dependent matrix, meaning we can't use the same setup (e.g., L & U) for each time step

Other complications

- Frequently the shape of the domain changes with each time step (or often). In such cases, not only does A change, but the grid must be recomputed
- Our example was a scalar equation (one PDE, one unknown). Much more common are systems (multiple PDEs, several unknowns at each spatial point). That is, much more complicated equations:

$$\frac{\partial^2 u}{\partial x^2}+\frac{1-\upsilon}{2}\left(\frac{\partial^2 u}{\partial y^2}+\frac{\partial^2 u}{\partial z^2}\right)+\frac{1+\upsilon}{2}\left(\frac{\partial^2 v}{\partial x \partial y}+\frac{\partial^2 w}{\partial x \partial z}\right)=f_1$$

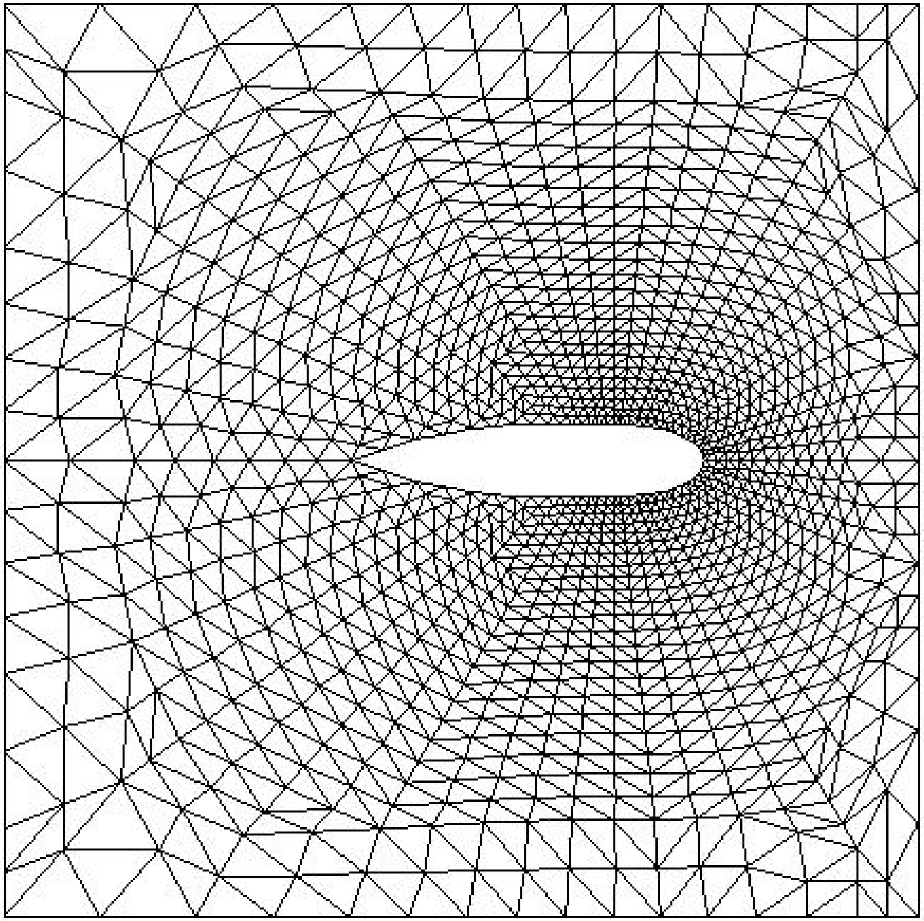
$$\frac{\partial^2 v}{\partial y^2}+\frac{1-\upsilon}{2}\left(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial z^2}\right)+\frac{1+\upsilon}{2}\left(\frac{\partial^2 u}{\partial x \partial y}+\frac{\partial^2 w}{\partial y \partial z}\right)=f_2$$

$$\frac{\partial^2 w}{\partial z^2}+\frac{1-\upsilon}{2}\left(\frac{\partial^2 w}{\partial x^2}+\frac{\partial^2 w}{\partial y^2}\right)+\frac{1+\upsilon}{2}\left(\frac{\partial^2 u}{\partial x \partial z}+\frac{\partial^2 v}{\partial y \partial z}\right)=f_3$$

Complications: complex grid geometry

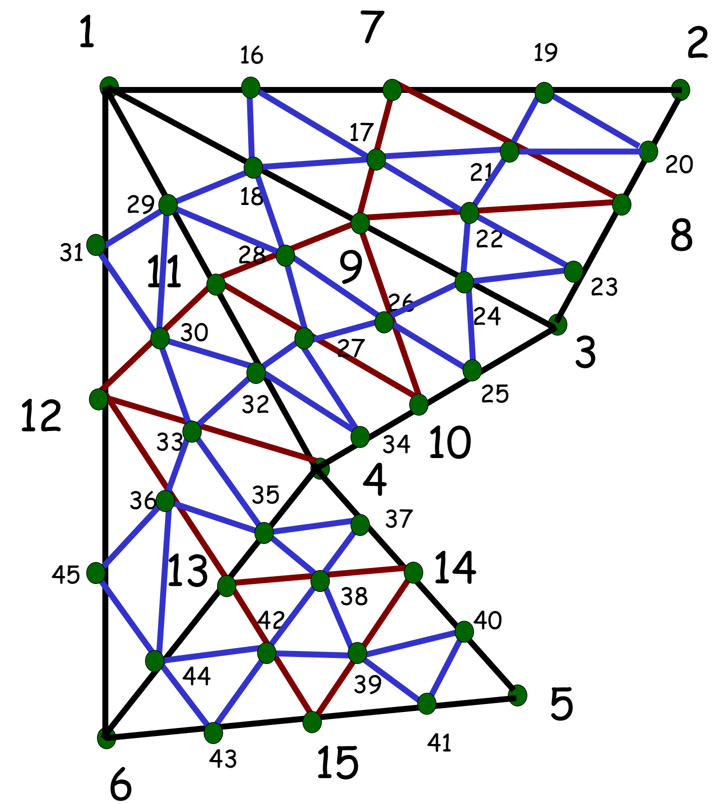
- Many problems cannot be posed simply on regular Cartesian grids, as they require grids upon or around irregularly shaped physical bodies.

- Example: a grid for computing flow around an airfoil:



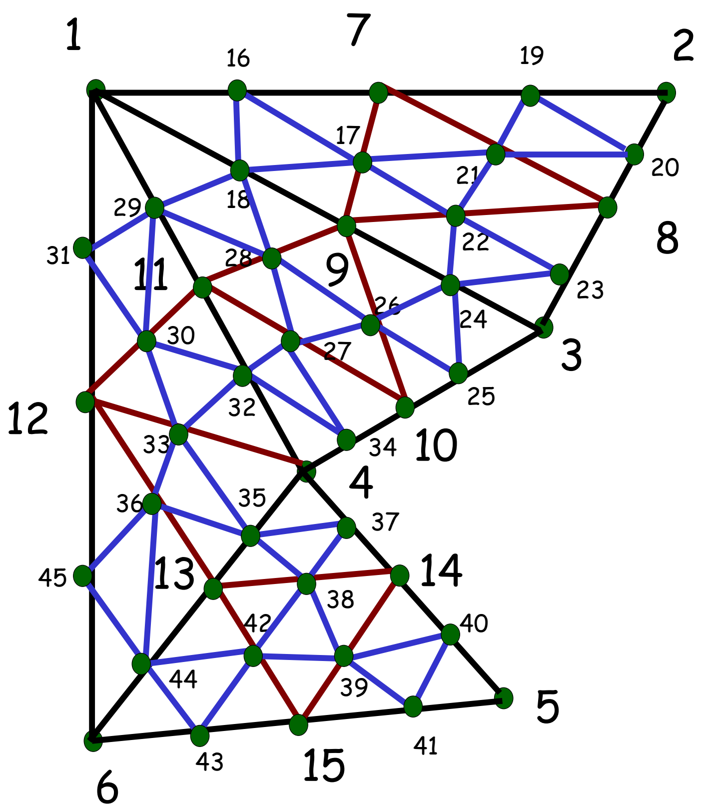
Complex grid geometry, con't

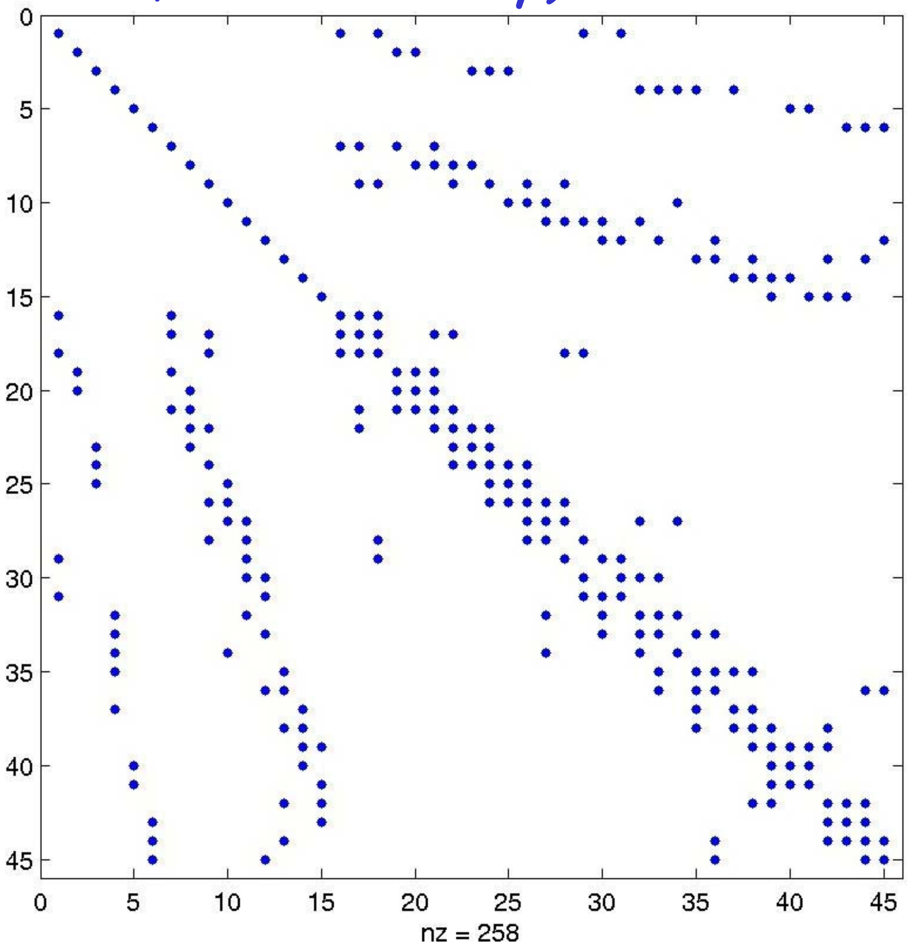
- Irregular grids are often generated by triangulations involving successive refinements
- Stencils reach near neighbor points, but they are spread far across the matrix
- Hence, problems on irregular grids don't have tightly banded matrices
- Example: point 18 connects directly to 1, 9, 16, 17, 28, 29



Complex grid geometry, con't

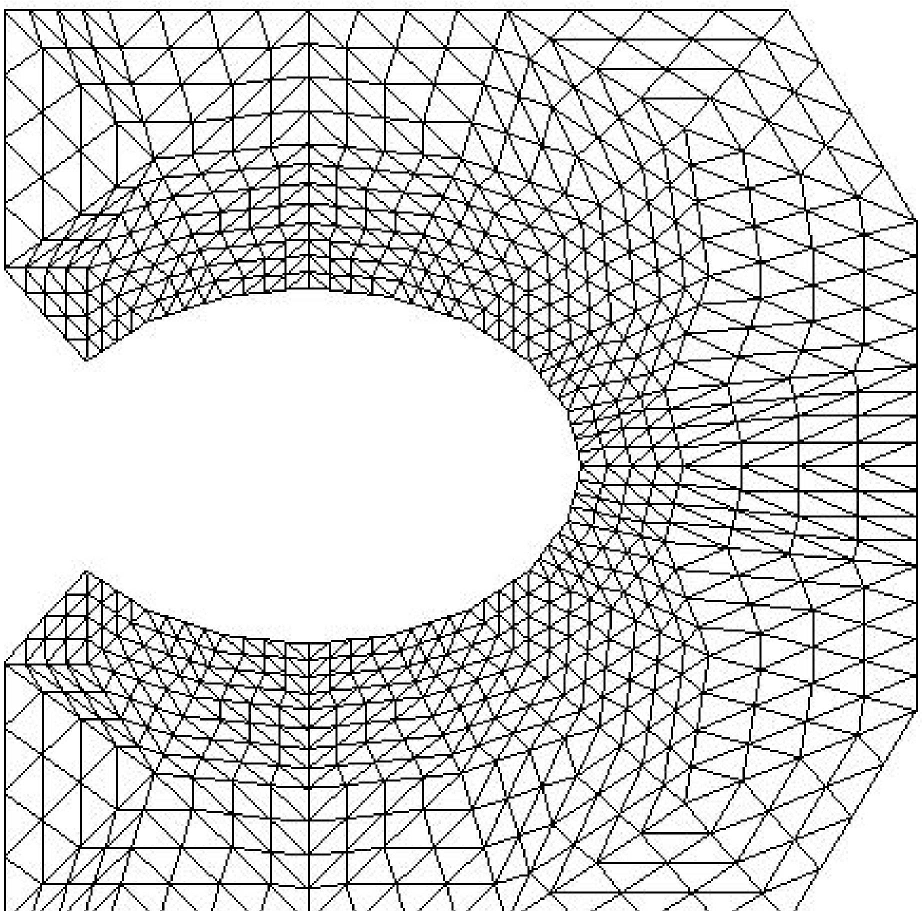
- Assuming closest neighbor connections, the grid on the left produces the matrix nonzero pattern shown (45x45 matrix, 258 nonzeros, 12% density)

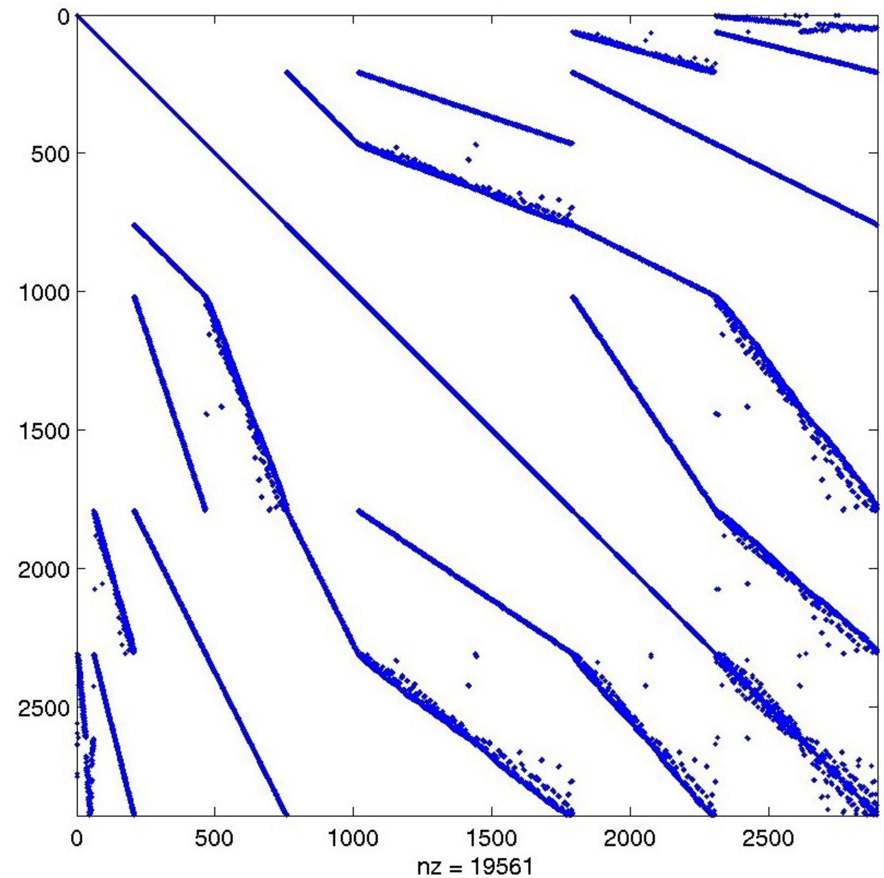




Complex grid geometry, con't

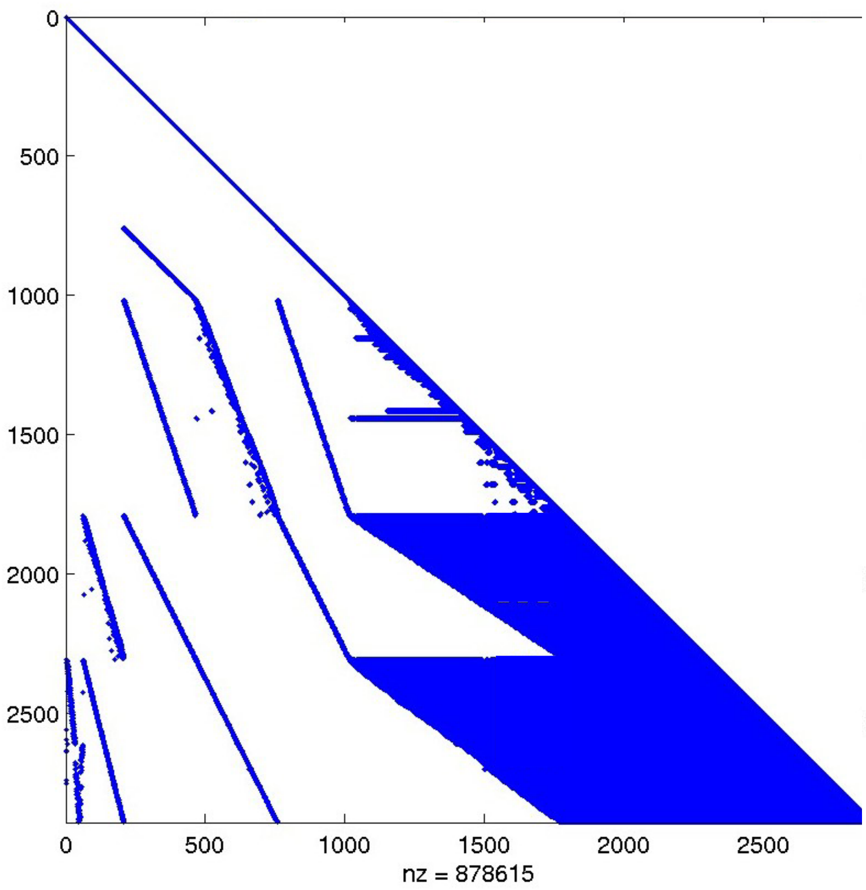
- Assuming closest neighbor connections, the grid on the left produces the matrix nonzero pattern shown (2889x2889 matrix, 19561 nonzeros, 0.2% density)

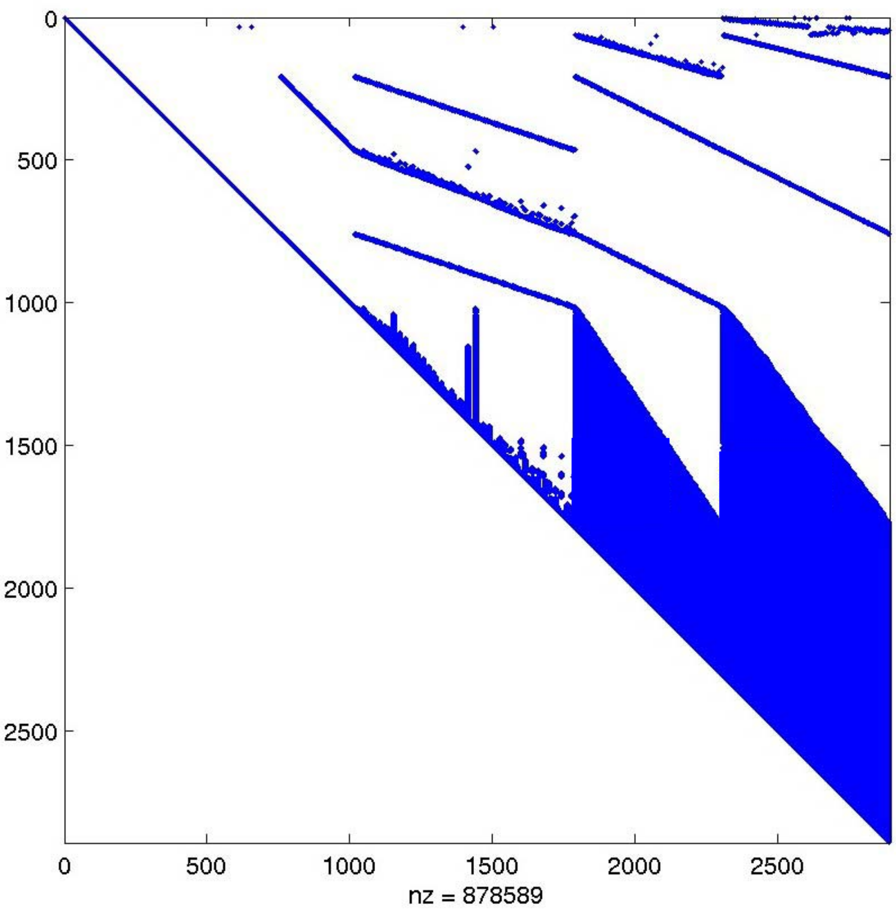




Now, how about LU factorization?

- The nonzero structures of L and U are:

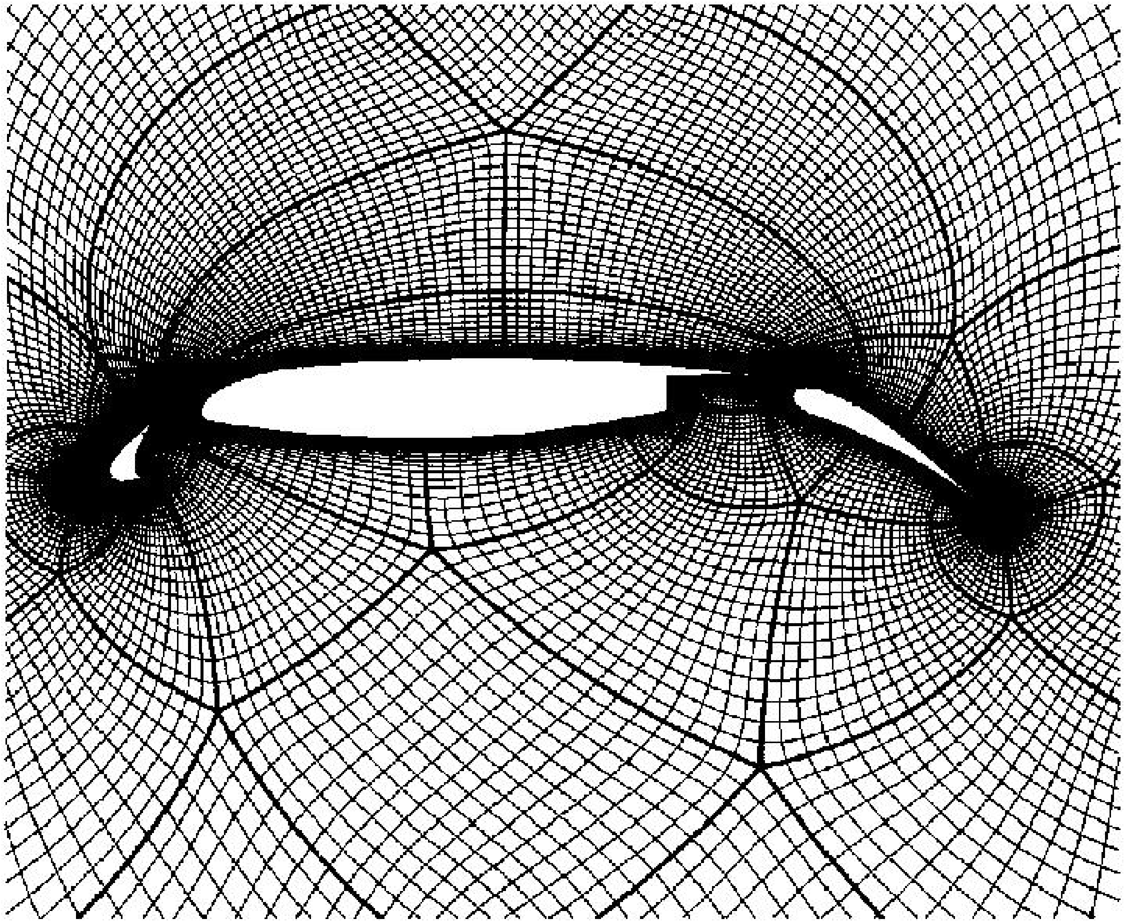




Semi-Structured-Grid System

- Allows more general grids
  - Grids that are mostly—but not entirely—structured
  - Example: block-structured grids

$$
\begin{bmatrix}
G_{11} & G_{12} & & & G_{15} & G_{16} & & G_{18} & \\
G_{21} & G_{22} & G_{23} & G_{24} & G_{25} & & G_{27} & G_{28} & \\
 & G_{32} & G_{33} & & & & G_{37} & G_{38} & \\
 & G_{42} & & G_{44} & G_{45} & & G_{47} & & \\
G_{51} & G_{52} & & G_{54} & G_{55} & G_{56} & & & G_{59} \\
G_{61} & & & & G_{65} & G_{66} & & G_{68} & G_{69} \\
 & G_{72} & G_{73} & G_{74} & & & G_{77} & & \\
G_{81} & G_{82} & G_{83} & & & G_{86} & & G_{88} & \\
 & & & & G_{95} & G_{96} & & & G_{99}
\end{bmatrix}
$$



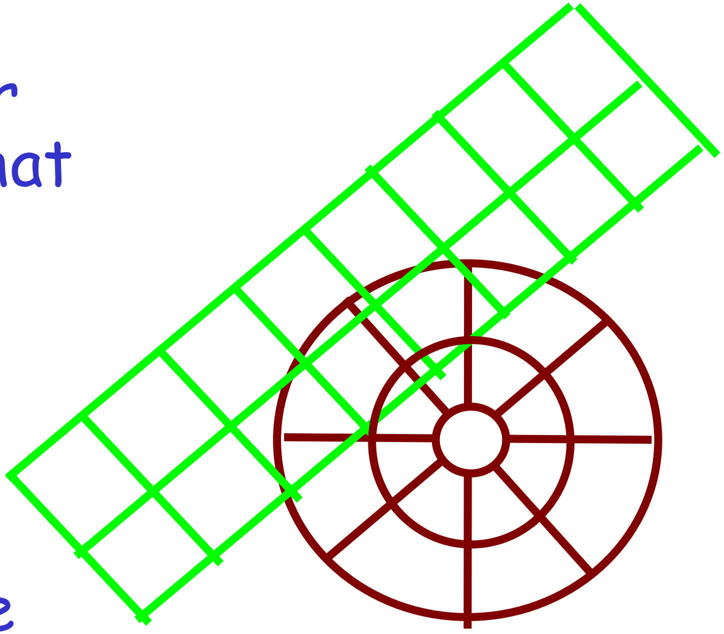
What solvers to use?

- Direct, i.e., Gaussian Elimination or LU?
- Iterative:
- Jacobi or Gauss-Seidel, *A=(D+L+U)*
  - Jacobi $x \leftarrow D^{-1}(L+U)x + D^{-1}b$
  - Gauss-Seidel $x \leftarrow (D+L)^{-1}Ux + (D+L)^{-1}b$
- Krylov, i.e., CG or GMRES
  - Required operations: $Ax, \quad x^T y$
- Multigrid

$$x^f \leftarrow \left(I + I_c^f \left(A^c\right)^{-1} I_f^c \left(b^f - A^f\right)\right) S^\upsilon x^f$$

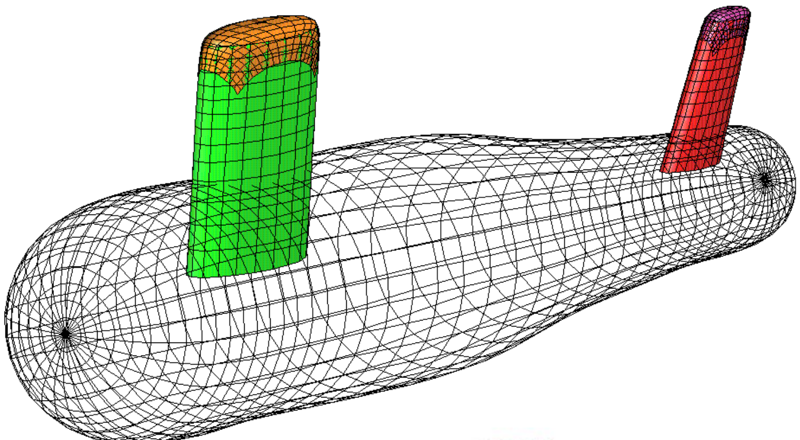
Handling complex geometries with overset grids

- Overset grids handles complex geometries by posing the problem on overlapping regular (logically rectangular) grids that conform to the geometry.
- The problem is solved on each grid.
- Solutions in the regions of overlap must be "adjudicated" somehow (it could be as simple as averaging, or extremely complex, depending on circumstances).

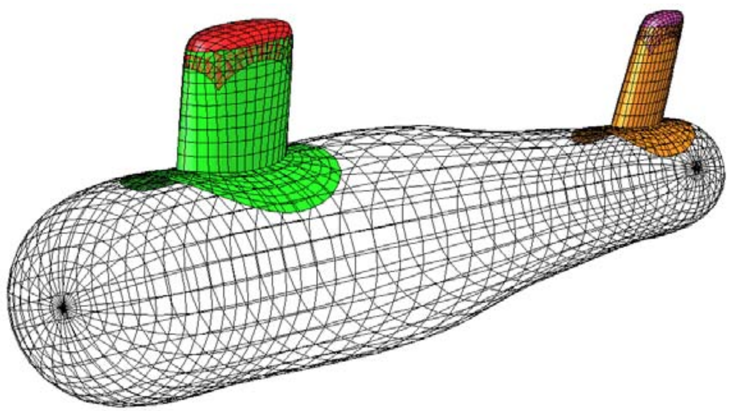


OVERTURE

The overset approach is based on component assembly

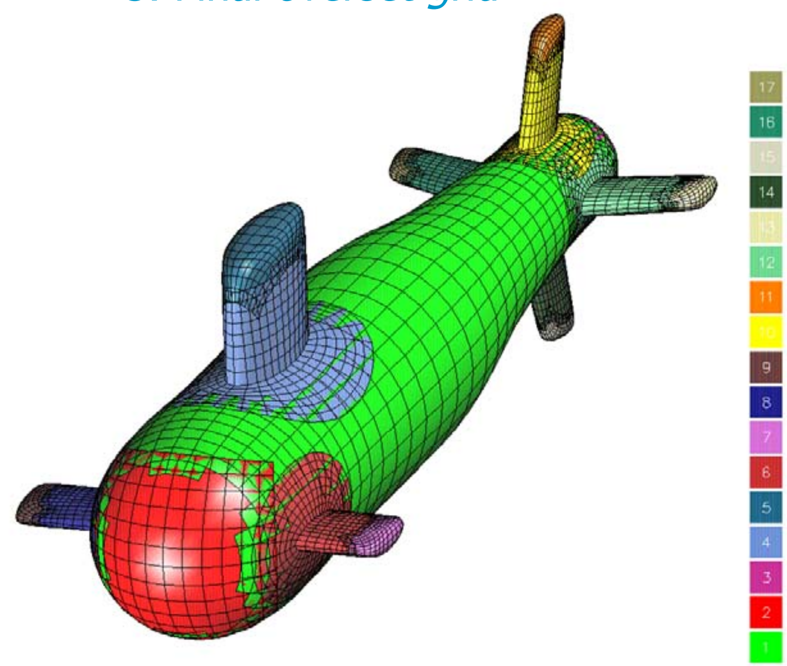


1. Components assembled

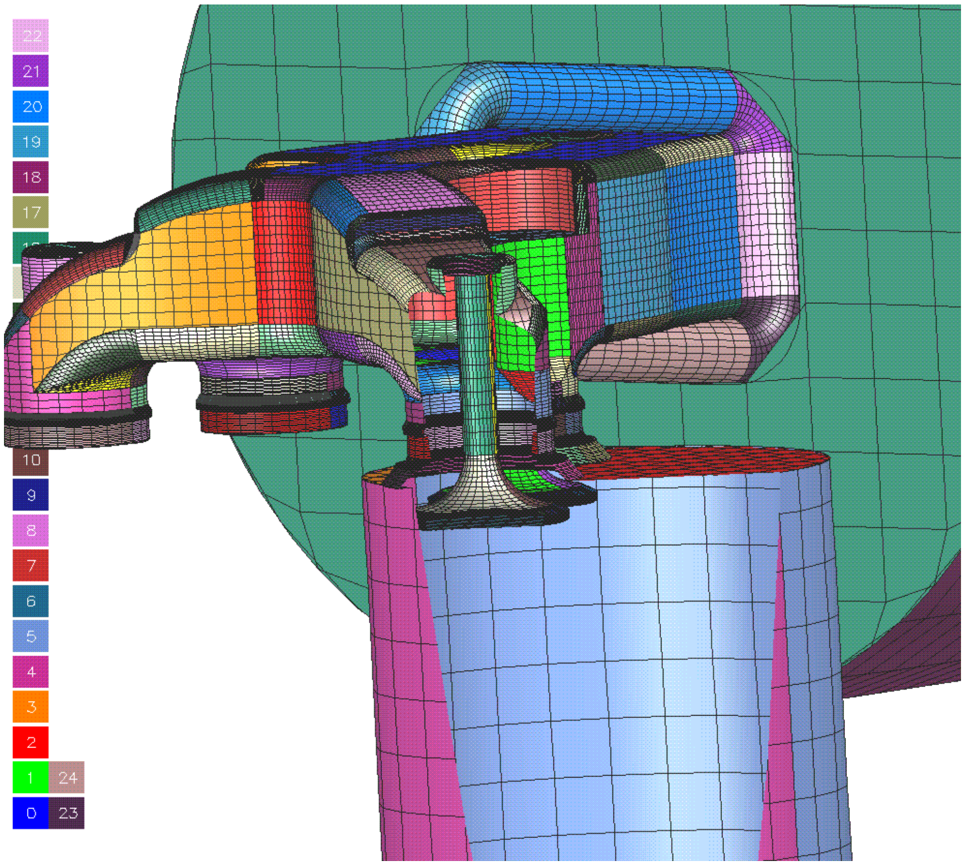


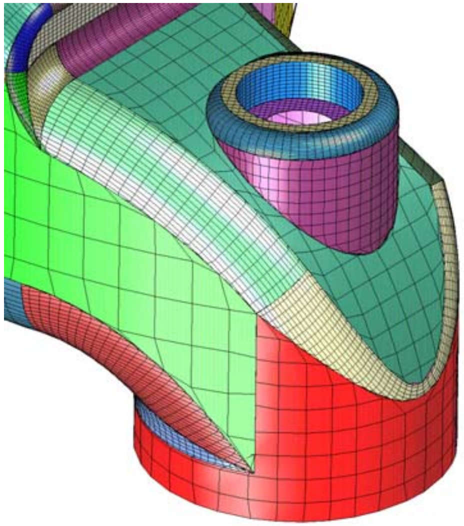
2. Intersections computed automatically; blended to submarine body surface

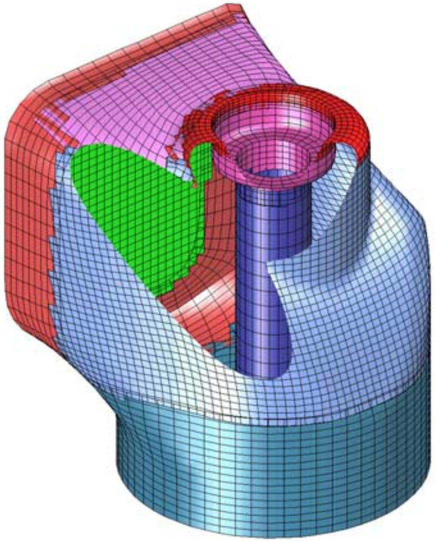
3. Final overset grid



For combustion simulations, grids are constructed from CAD data

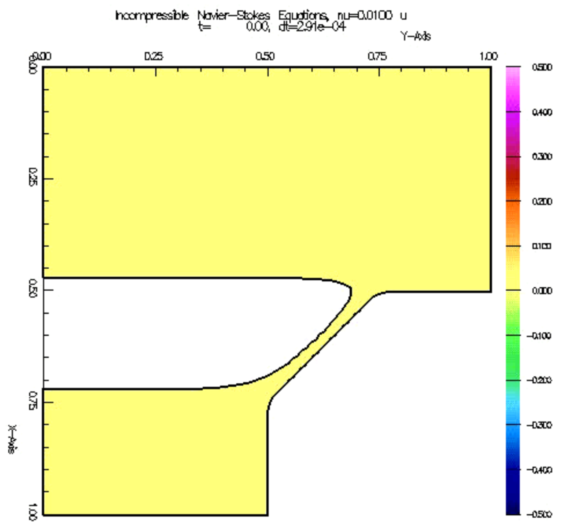


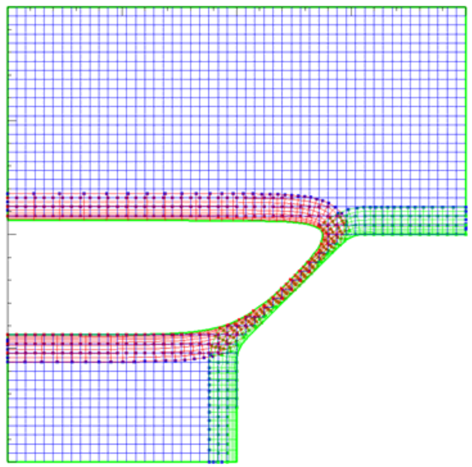




OVERTURE

supports overset grid technology for complex moving geometries

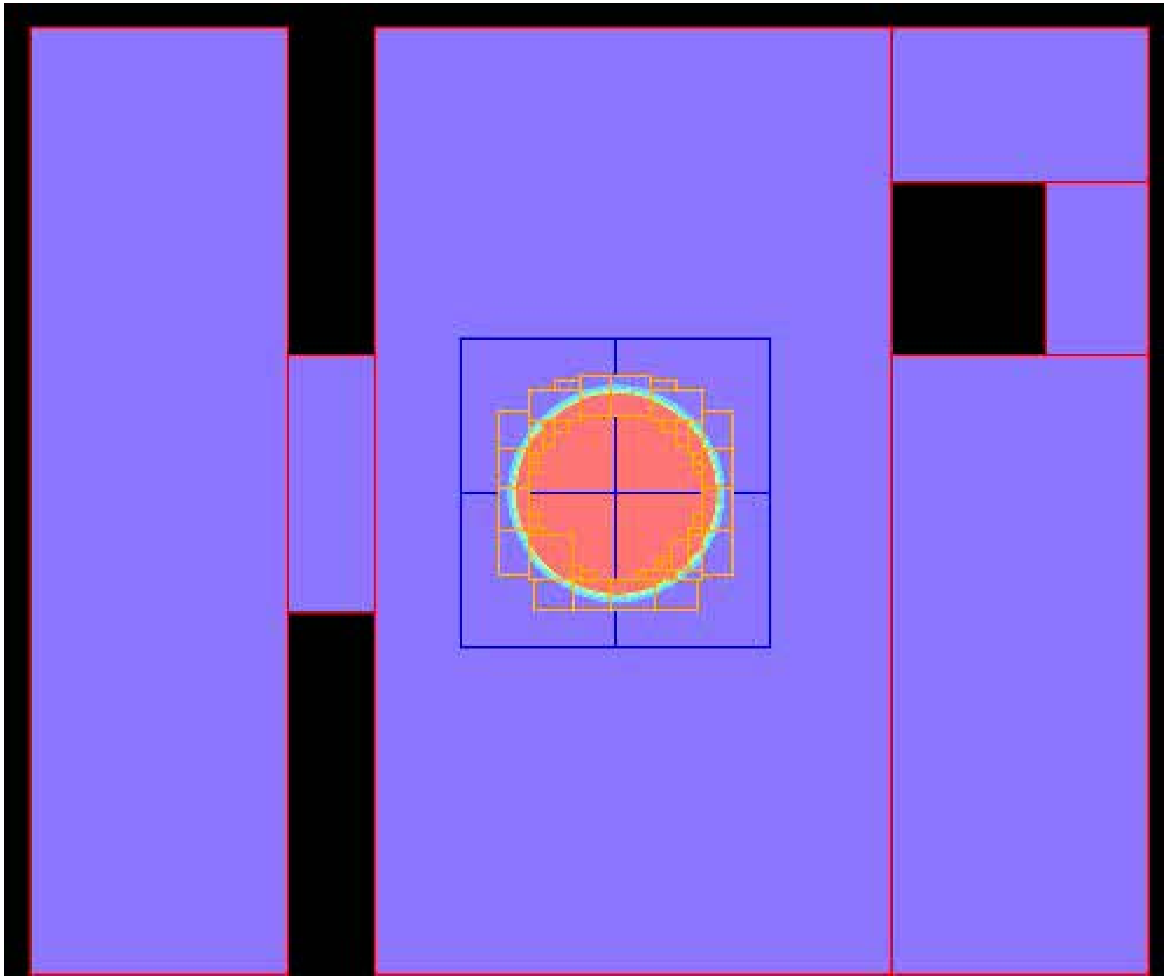




- Object-oriented tools for solving CFD and combustion problems in complex moving geometry
- Portable solution for serial and parallel environments using P++ array class
- Adaptive mesh refinement capabilities
- Finite Difference and Finite Volume technology
- Incompressible, Nearly incompressible and Compressible Flow solvers

SAMRAI focuses computational effort where it is needed

- Adaptively refine grid in vicinity of interesting behavior
- SAMRAI is an object-oriented code framework
- Parallelism is handled by the infrastructure, not the user

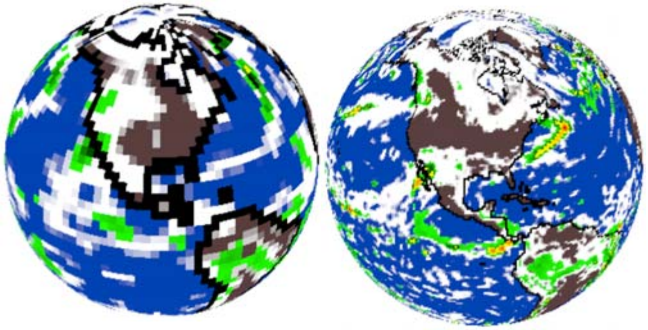


three levels of mesh resolution (4X):

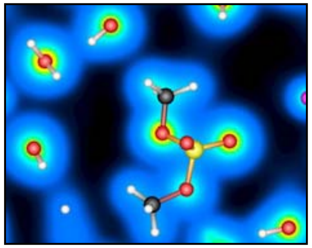
coarse
intermediate
fine

The need for parallelism

- To get accurate simulations, the physicists, chemists, and engineers demand ever increasing computing power:



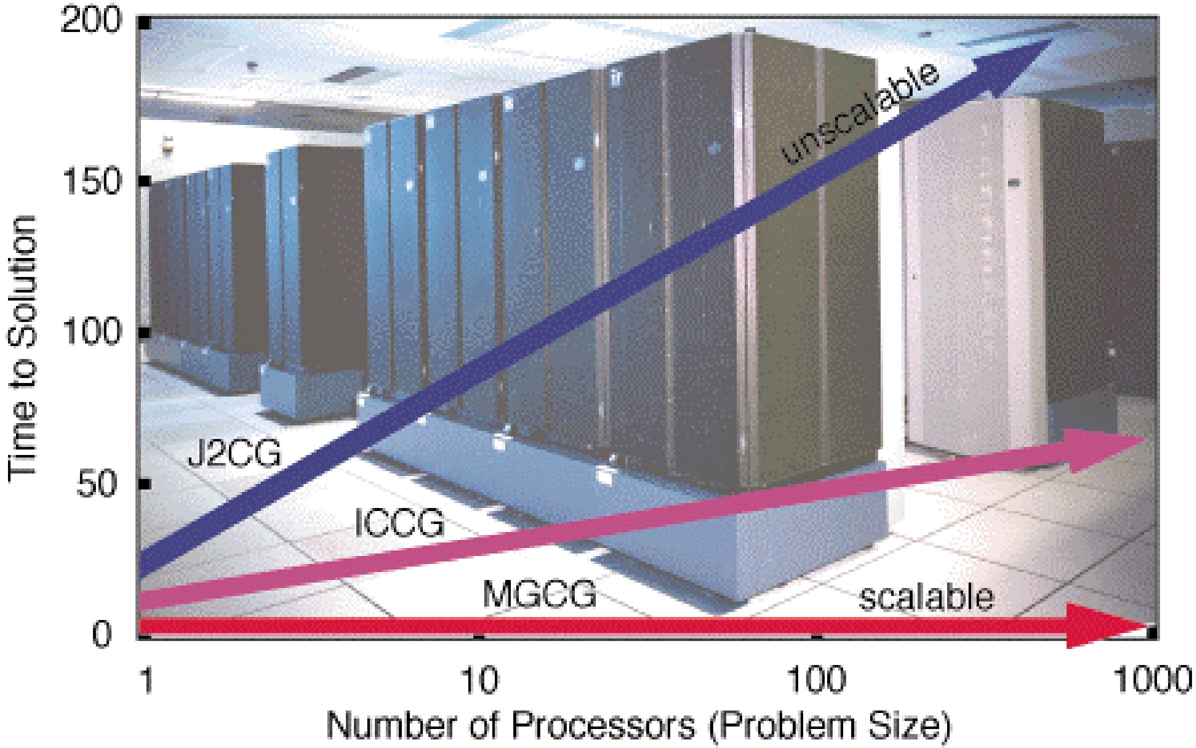
- Meterology- want samples spaced in the tens of miles, but over the entire globe! (1 every ten miles at surface, extend up 30 miles, gives 650 million gridpoints)



Molecular dynamics – at the atomic scale- want to calculate interaction of atoms!

- Material deformation and elasticity- need to solve equations on grids with ~10 M points every few nano- or pico- seconds

Scalability is a central issue for large-scale parallel computing

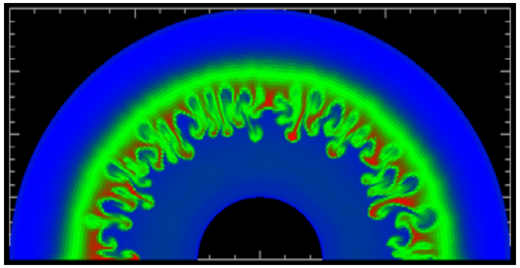


Linear solver convergence can be discussed independent of parallel computing, and is often overlooked as a key scalability issue.

LLNL's ASCI White is capable of 12.3 trillion operations per second

- ASCI White weighs 106 tons and covers 12,000 square feet of floor space (an area greater than that of two NBA basketball courts).
- It contains 8,192 microprocessors in 512 shared memory nodes.
- Each node contains 16 Power3-II CPUs built with IBM's latest semi-conductor technology (silicon-on-insulator and copper interconnects).
- Its 8 TB of memory is 125,000 times that of a 64-MB PC.
- 160 TB of storage in 7000 disk drives provides about 16,000 times the storage capacity of a PC with a 10-GB hard drive.

LLNL is a leader in scalable numerical algorithms R & D



LLNL multigrid solvers have sped up simulation codes 10X or more

Scalable algorithm

Procs	Size (M)	SMG
1	0.064	6
8	0.512	6
64	4.096	7
125	8.000	7
200	12.800	7

+

Scalable implementation

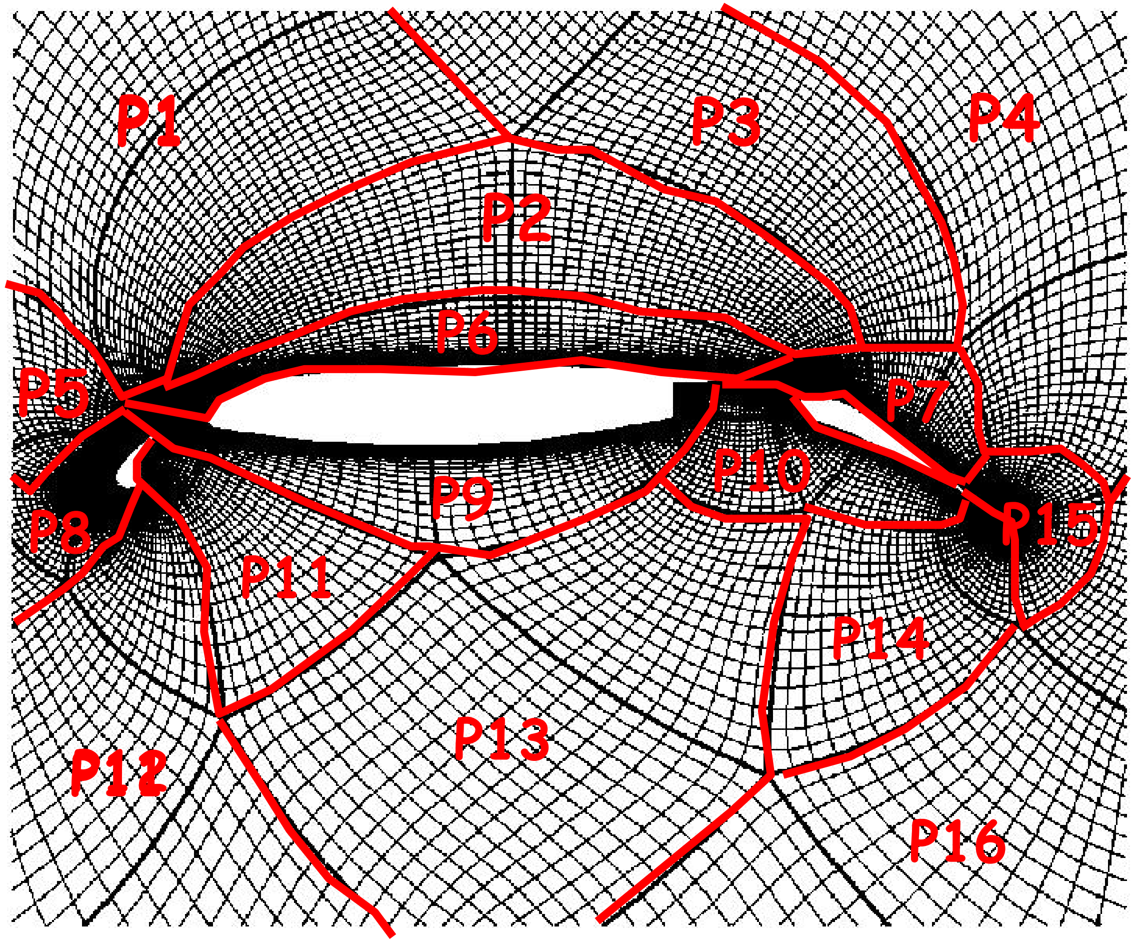
Procs	Size (M)	Seconds	Efficiency
1024	67.1	26	41%
2048	134.2	24	43%

= ***major impact on performance!***

For example: Algebraic multigrid solver for unstructured mesh problems enables ICF simulations on 1500+ processors

How do we divide up problems for multiple processors?

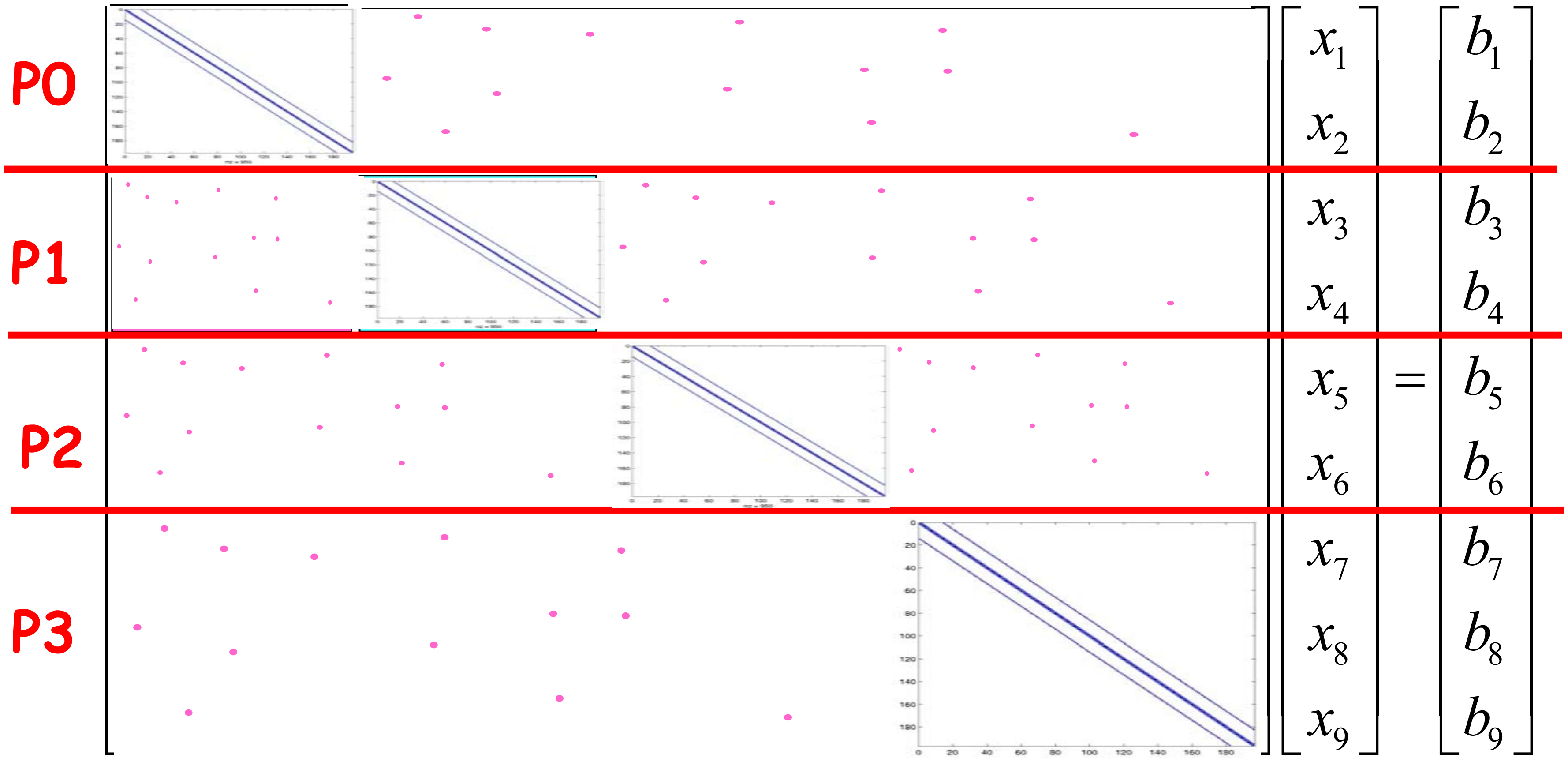
- Usually partition the domain
- Local connections
- Load balance



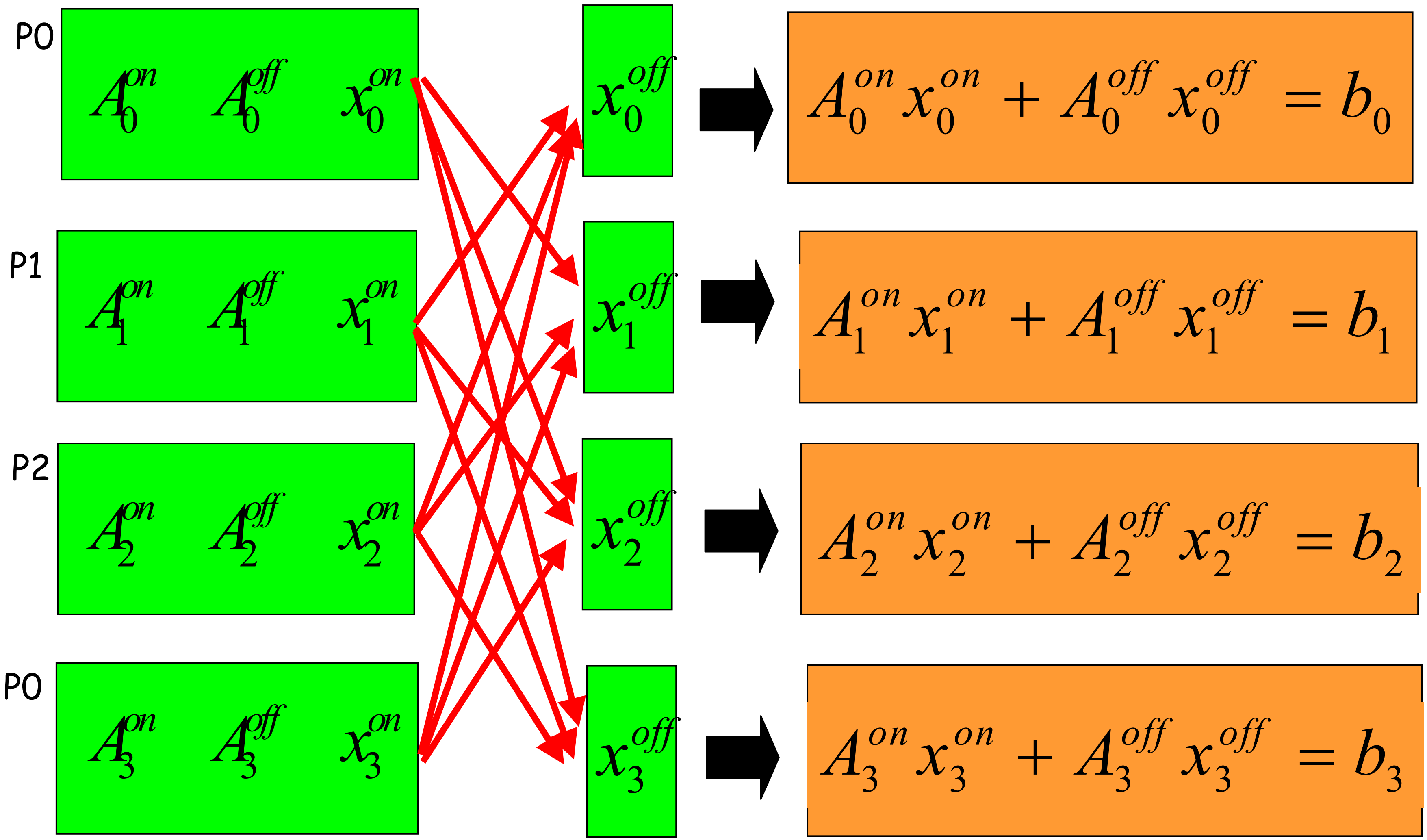
This partitions the matrix, vectors

= Connections for points local to the processor

= Connections to data stored off-processor



Performing the matrix-vector multiply Ax=b



Neutron Transport

3-d time-dependent Boltzmann equation for neutron transport

$$\frac{1}{v(E)}\frac{\partial\psi}{\partial t}+\Omega\bullet\nabla\psi+\sigma(r,E)\psi=$$

$$\int_0^{\infty}\int_{S^2}\sigma_s(r,\Omega'\bullet\Omega,E'\to E)\psi(r,\Omega',E')d\Omega' dE'+q$$

where

$$\begin{aligned}
\psi(r,\Omega,E,t) &= \text{flux or intensity}\\
r &= (x,y,z)\\
E,E' &= \text{energies}\\
\Omega,\Omega' &= \text{directions}\\
q(r,\Omega,E,t) &= \text{source}\\
v(E) &= \text{particle speed}\\
\sigma &= \text{total cross section}\\
\sigma_s &= \text{scattering cross section}
\end{aligned}$$

Neutron Transport

Discretization approaches

- "Multigroup" Energy discretization

$$0 \leq E_G < \cdots < E_g < E_{g-1} < \cdots < E_0$$

- Directional discretizations

S_N Method $$\int_{S^2} f(\Omega)\,d\Omega \approx \sum_i w_i f(\Omega_i)$$

P_N Method $$\psi(r,\Omega) \approx \sum_{n=0}^{N} \sum_{m=-n}^{n} \varphi_n^m(r) Y_n^m(\Omega)$$

- Spatial discretizations

d-differencing, finite element, subcell balance methods

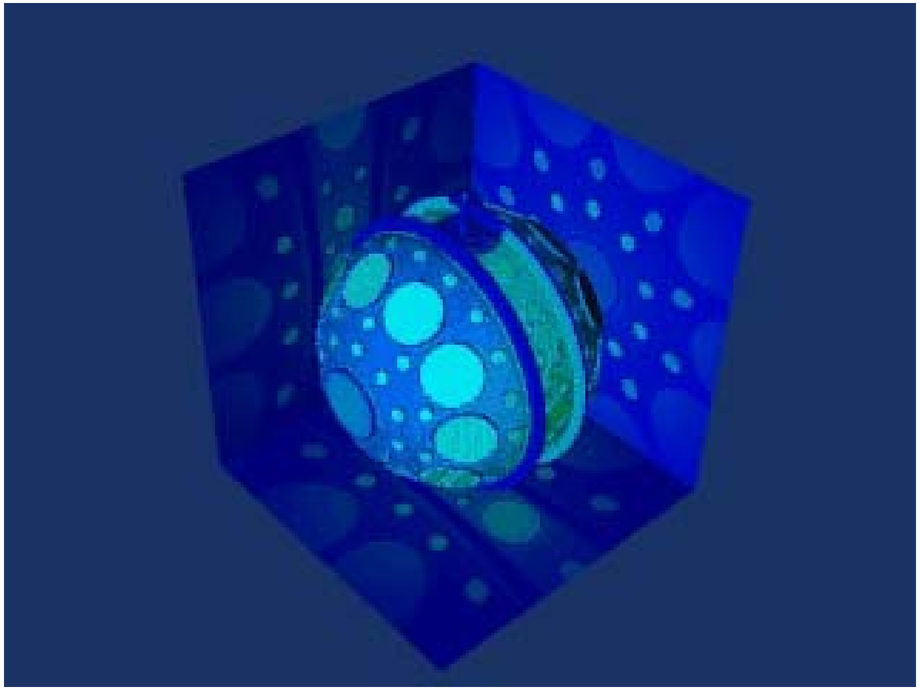
- Time discretization

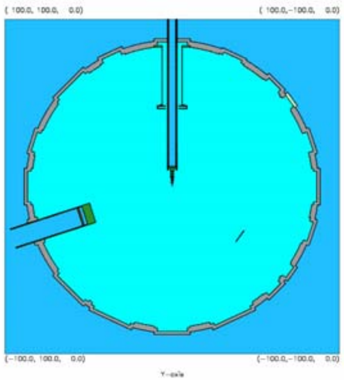
implicit timestepping coupled with operator splitting

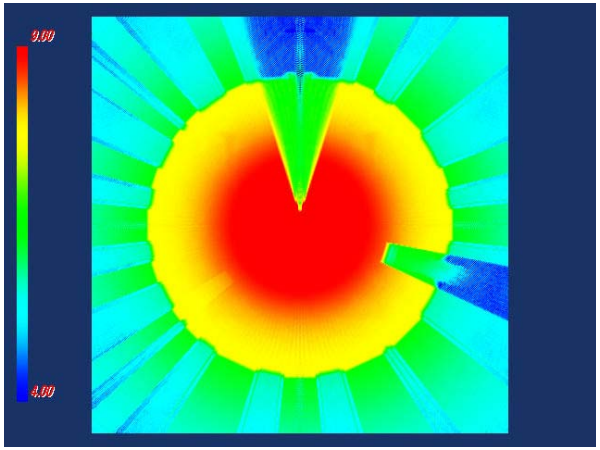
NEUTRON TRANSPORT

Shielding Calculation of the Nova Target Chamber using Ardra

- 15 billion unknowns
  - 23 energy groups
  - 160 million zones
  - P_1 approximation
  - first scatter point source
- BiCGSTAB iteration
- 3840 processors
- 27 hours



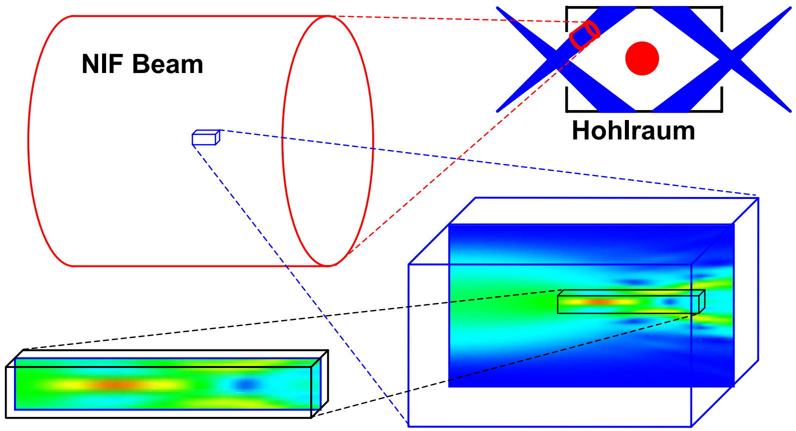




Ardra results: visualization of neutron scalar flux for highest energy group

Colors represent flux values: red highest, blue lowest

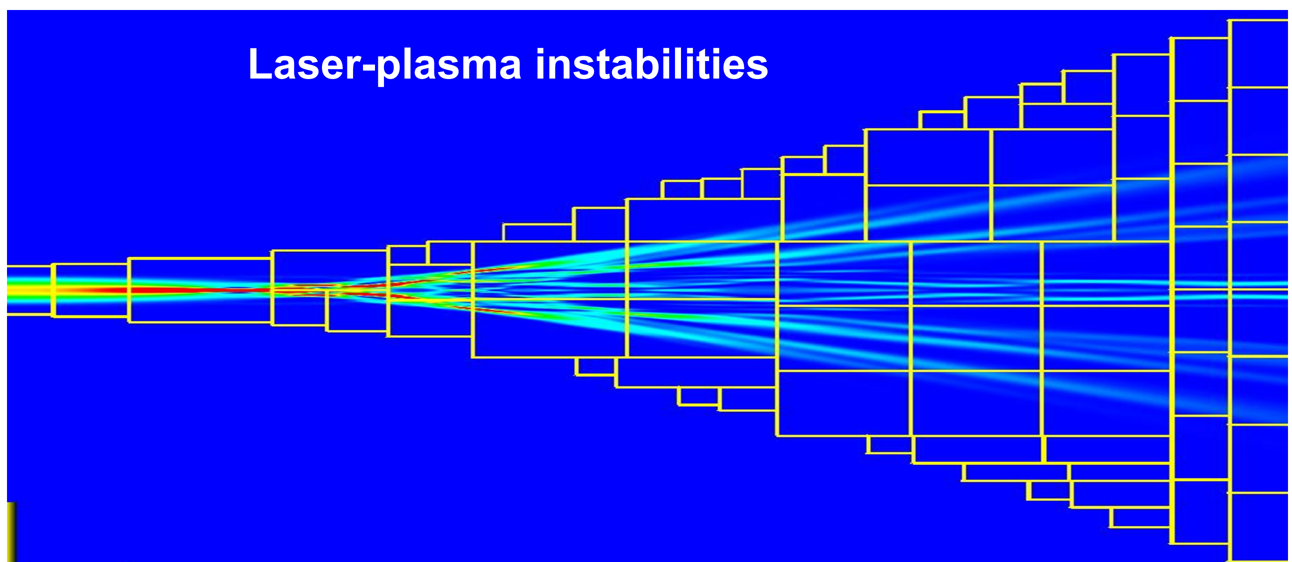
ALPS: Laser-plasma simulation using AMR



Adaptive Mesh Refinement (AMR) allows one to focus computational resources where they are most needed

ALPS simulation of light intensity in a filamented beam

512 x 512 coarse grid, refined 4 x 4



ALPS

Euler-Poisson model

$$\partial_t n_i + \nabla\cdot(n_i u_i) = 0$$

Ions

$$\partial_t(m_i n_i u_i) + \nabla\cdot(m_i n_i u_i \otimes u_i) + \nabla p_i = Zen_i E$$

$$(\gamma-1)^{-1}[\partial_t p_i + \nabla\cdot(p_i u)] + p_i \nabla\cdot u = 0 \qquad p_i \equiv nkT_i$$

$$\partial_t n_e + \nabla\cdot(n_e u_e) = 0$$

Electrons

$$\partial_t(m_e n_e u_e) + \nabla\cdot(m_e n_e u_e \otimes u_e) + \nabla p_e = -en_e E + F_p$$

$$(\gamma-1)^{-1}[\partial_t p_e + \nabla\cdot(p_e u)] + p_e \nabla\cdot u = 0 \qquad p_e \equiv nkT_e$$

$$\varepsilon_0 \nabla\cdot E = e(Zn_i - n_e) \qquad E = -\nabla\varphi$$

Field

$$-\left(\frac{\partial^2}{\partial t^2} - c^2\nabla^2\right)\vec{E}_0 = \frac{e^2 n_e}{\varepsilon_0 m_e}\vec{E}_0$$

Light

ALPS
Light model

From Maxwell's equations:

$$-\left(\frac{\partial^2}{\partial t^2} - c^2\nabla^2\right)\vec{E}_0 = \frac{e^2 n_e}{\varepsilon_0 m_e}\vec{E}_0 \quad \text{and} \quad \vec{E}_0 \equiv \vec{e}E_0(x)\exp\left[i\left(-\omega_0 t + \int^z k_0(z')dz'\right)\right] + \text{c.c.} \Rightarrow$$

$$\boxed{\left(c^2\frac{\partial^2}{\partial z^2} + 2ik_0c^2\frac{\partial}{\partial z} + ic^2k_0' + c^2\nabla_\perp^2 + 2i\omega_0\nu\right)E_0 = \frac{e^2 n_e}{\varepsilon_0 m_e}E_0}$$

$$c^2k_0^2 = \omega_0^2 - \frac{e^2\overline{n}_e}{\varepsilon_0 m_e}$$

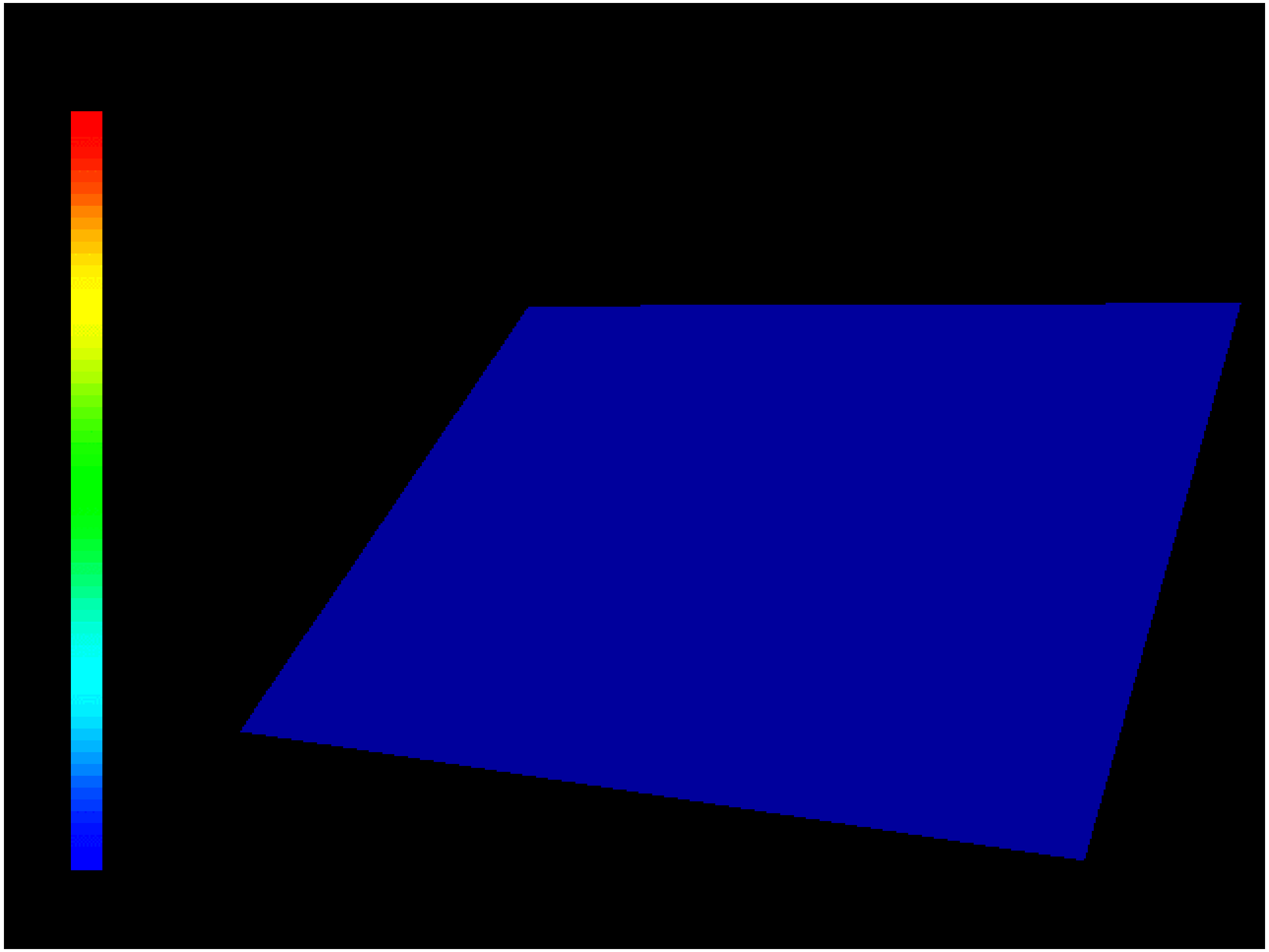
$$n_e \equiv \overline{n}_e + \delta n_e$$

Refraction: $$\left(\frac{\partial}{\partial z} + \frac{ie^2\delta n_e}{2c^2k_0\varepsilon_0 m_e}\right)E_0 = 0$$

Absorption: $$\left(\frac{\partial}{\partial z} + \frac{k_0\nu}{\omega_0}\right)E_0 = 0$$

Diffraction: $$\left[\frac{\partial}{\partial z} + ik_0 - i\left(\nabla_\perp^2 + k_0^2\right)^{1/2}\right]E_0 = 0$$

ALPS simulation



The Big kid's Playground

- So, if you know how to solve Ax=b, if you know how to partition data, and if you know a spot about differential equations and numerical analysis, you can go from here

$$\frac{\partial^2 u}{\partial x^2}+\frac{1-\upsilon}{2}\left(\frac{\partial^2 u}{\partial y^2}+\frac{\partial^2 u}{\partial z^2}\right)+\frac{1+\upsilon}{2}\left(\frac{\partial^2 v}{\partial x\partial y}+\frac{\partial^2 w}{\partial x\partial z}\right)=f_1$$

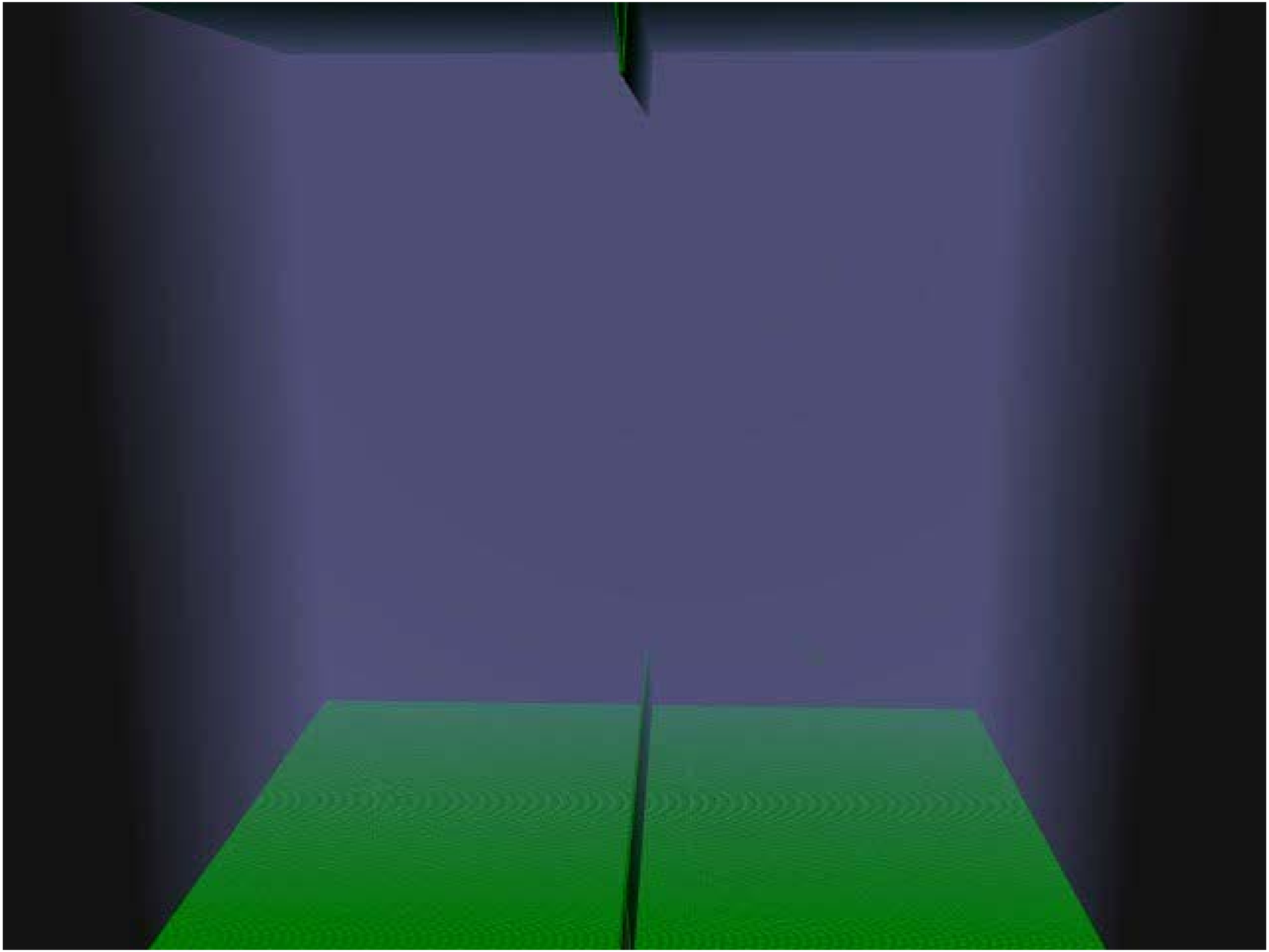
$$\frac{\partial^2 v}{\partial y^2}+\frac{1-\upsilon}{2}\left(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial z^2}\right)+\frac{1+\upsilon}{2}\left(\frac{\partial^2 u}{\partial x\partial y}+\frac{\partial^2 w}{\partial y\partial z}\right)=f_2$$

$$\frac{\partial^2 w}{\partial z^2}+\frac{1-\upsilon}{2}\left(\frac{\partial^2 w}{\partial x^2}+\frac{\partial^2 w}{\partial y^2}\right)+\frac{1+\upsilon}{2}\left(\frac{\partial^2 u}{\partial x\partial z}+\frac{\partial^2 v}{\partial y\partial z}\right)=f_3$$

to

$$\begin{bmatrix}
G_{11} & G_{12} & & & G_{15} & G_{16} & & G_{18} & \\
G_{21} & G_{22} & G_{23} & G_{24} & G_{25} & & G_{27} & G_{28} & \\
 & G_{32} & G_{33} & & & & G_{37} & G_{38} & \\
 & G_{42} & & G_{44} & G_{45} & & G_{47} & & \\
G_{51} & G_{52} & & G_{54} & G_{55} & G_{56} & & & G_{59} \\
G_{61} & & & & G_{65} & G_{66} & & G_{68} & G_{69} \\
 & G_{72} & G_{73} & G_{74} & & & G_{77} & & \\
G_{81} & G_{82} & G_{83} & & & G_{86} & & G_{88} & \\
 & & & & G_{95} & G_{96} & & & G_{99}
\end{bmatrix}
\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \end{bmatrix}
=
\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \\ b_8 \\ b_9 \end{bmatrix}$$

to…



LLNL's University Relations Program supports student internships

Students participate in computational science research projects under leading principal investigators

The ISCR alone hosted 55 students and 20 faculty during Summer 2000 through a variety of auspices

- ASCI Alliances
- UCDRD "mini-grants"
- LLNL student fellowships

- ASCI Computer Science Institute
- DOE Computational Science Graduate Fellows
- DOE HPC Computer Science Graduate Fellows
- Internships in Terascale Simulation Technology
- National Physical Science Consortium
- *and more!*